

# Instituto Politécnico Nacional

## Escuela Superior de Cómputo



## Estructuras de Datos

### *Tema 04:* TAD Lista

M. en C. Edgardo Adrián Franco Martínez

<http://www.eafranco.com>

[edfrancom@ipn.mx](mailto:edfrancom@ipn.mx)

[@edfrancom](https://twitter.com/edfrancom) [f edgardoadrianfrancom](https://www.facebook.com/edgardoadrianfrancom)





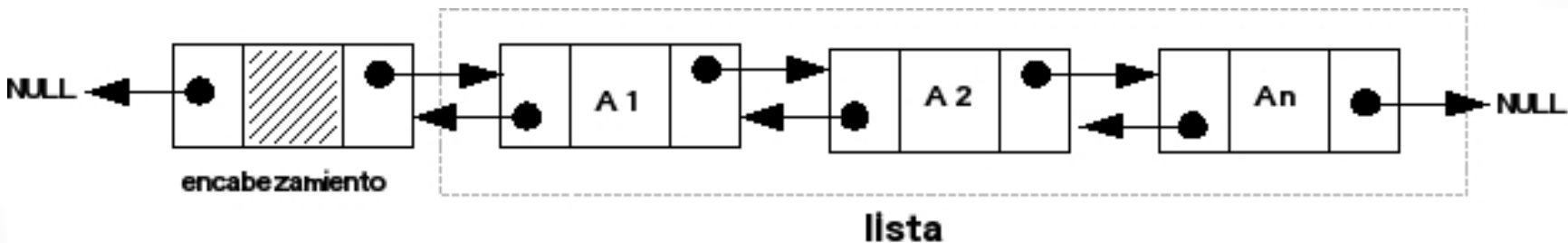
# Contenido

- TAD Lista
- Descripción del TAD Lista
- Especificación del TAD Lista
  - Cabecera
  - Definición
  - Operaciones
  - Observaciones
- Tipos de listas
  - Listas simplemente ligadas
  - Listas doblemente ligadas
- Aplicaciones del TAD Lista
- Modalidades de listas
  - Pilas como listas
  - Colas como listas
  - Otras modalidades de la lista



# TAD Lista

- Las **listas** son la generalización de los dos TAD's anteriores (Pila y Cola): mientras que en una pila y en una cola las operaciones sólo afectan a un extremo de la secuencia, en una lista se puede **insertar un elemento en cualquier posición**, y **borrar y consultar cualquier elemento**.





# Descripción del TAD Lista

- Una lista es:
  - Una *colección de 0 o mas elementos*
    - Si la lista no tiene elementos, se dice que esta vacía
  - En una lista, todos los elementos son de un mismo tipo
- Son estructuras lineales, i.e.
  - Sus elementos están colocados uno detrás de otro
  - Cada elemento de una lista se conoce con el nombre de **NODO**.
- Las listas
  - Son mucho más flexibles que los arreglos
  - Permiten trabajo “dinámico” con un grupo de elementos





- Una lista es una colección de elementos ordenada de acuerdo a las posiciones de éstos (secuencia, relación predecesor-sucesor).

$$\begin{array}{c} \text{primer elemento} \\ \downarrow \\ L = \langle a_1, a_2, \dots, a_n \rangle \\ \uparrow \\ \text{último elemento} \end{array}$$

❖  $a_i \in L, i=1, \dots, n$  ( $n$  es la *longitud* de la lista)

❖  $n=0 \Rightarrow$  lista *vacía*

❖ Caracterización importante: los elementos pueden insertarse o eliminarse en cualquier posición de una lista



# Especificación del TAD Lista

## Cabecera

- **Nombre:** LISTA (*LIST*)
- **Lista de operaciones:**
  - **Operaciones de construcción**
    - **Inicializar (Initialize):** Recibe una lista *L* y la inicializa para su trabajo normal.
    - **Eliminar (Destroy):** Recibe una lista *L* y la libera completamente.
  - **Operaciones de posicionamiento y búsqueda**
    - **Fin (Final):** Recibe una lista *L* y regresa la **posición** del final
    - **Primero (First):** Recibe una lista *L* y regresa la **posición** del primero
    - **Siguiente (Following):** Recibe una lista *L* y una **posición** *p*, regresa la posición siguiente a *p*.
    - **Anterior (Previous):** Recibe una lista *L* y una **posición** *p*, regresa la posición anterior a *p*.
    - **Buscar (Search):** Recibe una lista *L* y un **elemento** *e*, regresa la posición que coincida exactamente con el elemento *e*.



- **Operación de consulta**

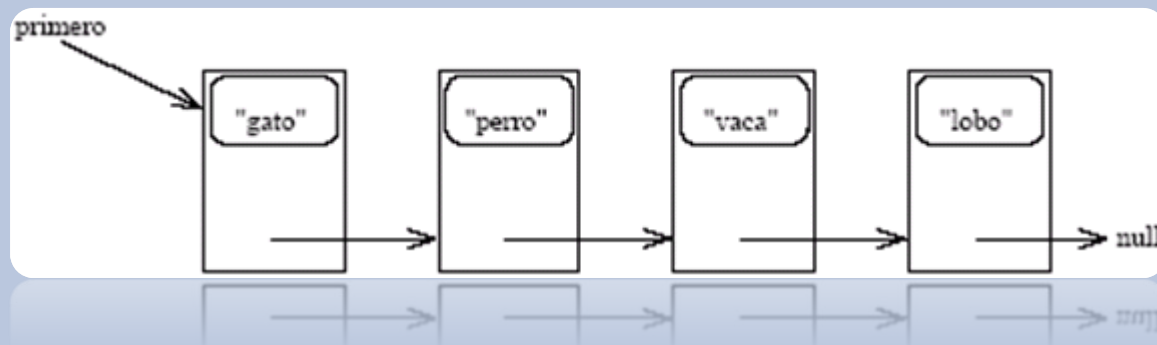
- **Posición (Position):** Recibe una lista **L** y una posición **p**, devuelve el elemento en dicha posición.
- **Validar posición (Validate Position):** Recibe una lista **L** y una posición **p**, devuelve verdadero en caso de que en la lista **L**, **p** sea una posición valida (no nula).
- **Elemento (Element):** Recibe una lista **L** y un número de elemento de 1 al tamaño de la lista y retorna al elemento de esa posición partiendo del frente de la lista como el elemento 1 hacia atrás.
- **Elemento posición (Element Position):** Recibe una lista y un número de elemento de 1 al tamaño de la lista y retorna la posición del elemento en ese número desde el frente de la lista.
- **Tamaño (Size):** Recibe una lista **L** y devuelve el tamaño de la lista
- **Vacía (Empty):** Recibe una lista **L** y devuelve verdadero en caso de que la lista **L** este vacía.

- **Operaciones de modificación**

- **Inserta (Insert):** Recibe una lista **L**, un elemento **e**, una posición **p** y un valor booleano **b** e inserta al elemento **e** en la lista **L** enfrente de **p** si **b** es verdadero o atrás de **p** en caso de que **b** sea falso.
- **Agregar (Add):** Recibe una lista **L** y un elemento **e**, se inserta al elemento **e** al final de la lista **L**.
- **Remover (Remove):** Recibe una lista **L** y una posición **p**, y elimina al elemento en la posición **p** de la lista.
- **Sustituir (Replace):** Recibe una lista **L**, un elemento **e**, una posición **p** y sustituye al elemento ubicado en **p** por **e**.

# Definición

- Una lista L es un conjunto de elementos del mismo tipo que:
  - O bien es vacío, en cuyo caso se denomina lista vacía.
  - O bien puede distinguirse un elemento, llamado cabeza o primero, y el resto de los elementos constituyen una lista L', denominada resto de la lista original.



- Consiste en una secuencia de nodos, en los que se guardan elementos y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales o arreglos es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.



# Operaciones



- **Inicializar (Initialize):** *recibe <- lista(L);*
  - **Initialize (L)**
  - **Efecto:** Recibe una lista L y la inicializa para su trabajo normal.
- **Eliminar (Destroy):** *recibe <- lista(L);*
  - **Destroy (L)**
  - **Efecto:** Recibe una lista L y la libera completamente
- **Fin(Final):** *recibe <- lista(L); retorna -> posición*
  - **p=Final (L)**
  - **Efecto:** Recibe una lista L y retorna la posición del elemento al final de esta.
- **Primero (First):** *recibe <- lista (L); retorna -> posición*
  - **p=First (L)**
  - **Efecto:** Recibe una lista L y devuelve la posición del elemento al inicio de esta.



- **Siguiente (Following):** *recibe* <- *lista (L), posición(P);*  
*retorna* -> *posición*
  - **p=Following (L,P)**
  - **Efecto:** Recibe una lista L, una posición P y devuelve la posición del elemento siguiente de P.
  - **Requerimientos:** La lista L es no vacía y la posición P es una posición válida.
- **Anterior (Previous):** *recibe* <- *lista (L), posición(P);*  
*retorna* -> *posición*
  - **p=Previous (L,P)**
  - **Efecto:** Recibe una lista L, una posición P y devuelve la posición del elemento anterior a P.
  - **Requerimientos:** La lista L es no vacía y la posición P es una posición válida.
- **Buscar (Search):** *recibe* <- *lista (L), elemento (e);*  
*retorna* -> *posición*
  - **p=Search (L,e)**
  - **Efecto:** Recibe una lista L y un elemento e, devuelve la posición del elemento que coincida exactamente con e.



- **Posición (Position):** *recibe* <- lista (L), posición(P); *retorna* -> elemento
  - **e=Position (L,P)**
  - **Efecto:** Recibe una lista L, una posición P y devuelve el elemento en dicha posición.
  - **Requerimientos:** La lista L es no vacía y la posición P es una posición válida.
- **Validar Posición (Validate Position):** *recibe* <- lista (L), posición(P); *retorna* -> boolean
  - **b=Position (L,P)**
  - **Efecto:** Recibe una lista L, una posición P y devuelve TRUE si la posición es una posición P válida en la lista L y FALSE en caso contrario.
- **Elemento(Element):** *recibe* <- lista (L); *recibe* <- índice(n); *retorna* -> elemento
  - **e=Element (L,n);**
  - **Efecto:** Recibe una lista y un índice (entre 1 y el tamaño de la lista) y devuelve el elemento que se encuentra en la lista en ese índice partiendo del frente de este =1 hacia atrás.
  - **Excepción:** Si la cola está vacía o el índice se encuentra fuera del tamaño de la lista se produce error.
- **Elemento Posición (Element Position):** *recibe* <- lista (L); *recibe* <- índice(n); *retorna* -> posición
  - **p=Element (L,n);**
  - **Efecto:** Recibe una lista y un índice (entre 1 y el tamaño de la lista) y devuelve la posición del elemento que se encuentra en la lista en ese índice partiendo del frente de este =1 hacia atrás.
  - **Excepción:** Si la cola está vacía o el índice se encuentra fuera del tamaño de la lista se retorna una posición inválida.



- **Tamaño (Size):** *recibe* <- lista (L), *retorna* -> tamaño
  - *n=Size (L)*
  - **Efecto:** Recibe una lista L y devuelve el tamaño de la lista.
- **Vacía (Empty):** *recibe* <- lista (L) *retorna* -> boolean
  - *b=Empty (L)*
  - **Efecto:** Recibe una lista L y devuelve TRUE en caso de que la lista este vacía y FALSE en caso contrario
- **Inserta (Insert):** *recibe* <- lista (L), *posición* (P), *elemento* (e), *boolean* (b);
  - *Insert (L,P,e,b)*
  - **Efecto:** Recibe una lista L, una posición P, un elemento e y un valor booleano; el elemento e deberá agregarse en la posición anterior de P si b es verdadero y en la posición siguiente de P en caso contrario.
  - **Requerimientos:** La posición P es una posición valida, si P es no valida o NULL, se insertará a e al frente de la lista.



- **Agregar (Add):** *recibe <- lista (L), elemento (e)*
  - **Add (L,e)**
  - **Efecto:** Recibe una lista L y un elemento e, se agrega a e al final de la lista L.
- **Remove (Remove):** *recibe <- lista (L), posición(P)*
  - **Remove (L,P)**
  - **Efecto:** Recibe una lista L y una posición P, el elemento en la posición P será removido.
  - **Requerimientos:** La lista L es no vacía y la posición P es una posición válida.
- **Sustituir (Replace):** *recibe <- lista (L), posición(P), elemento (e)*
  - **Replace (L,P,e)**
  - **Efecto:** Recibe una lista L, una posición P y un elemento e, el elemento en la posición P será sustituido por e
  - **Requerimientos:** La lista L es no vacía y la posición P es una posición válida.

# Observaciones

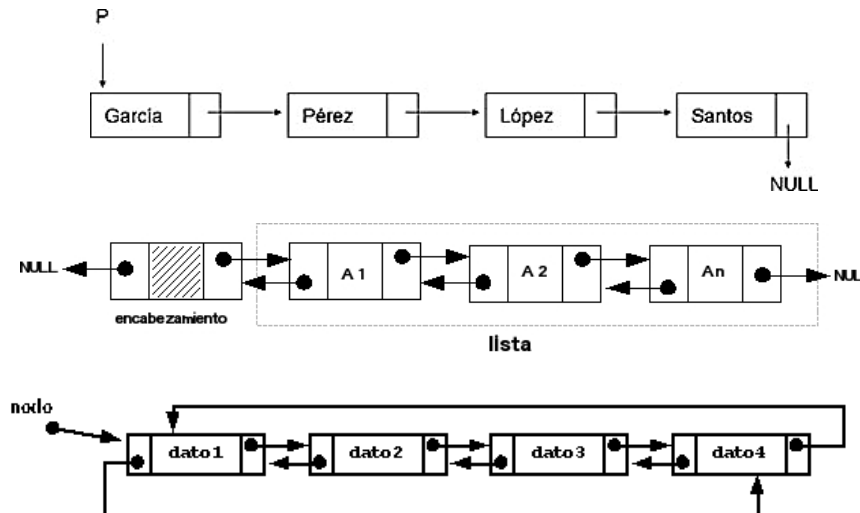
- Al remover un elemento de una lista de un solo elemento esta queda vacía
  - Una vez vacía, no se pueden “Sustituir o preguntar por elementos de una posición”.
- Antes de consultar un posición o usarla para llamar a alguna operación sobre la lista, se deberá de validar si esta posición es valida para la lista L.
- Los valores del TAD Lista son listas de elementos del tipo **Elemento**.
- Las posiciones de los elementos de la lista y la posición final de la lista son del tipo **Posición**.
- Las listas son **mutables**: Puede verse como Pila, Cola, etc. depende las operaciones y forma de manejarla.





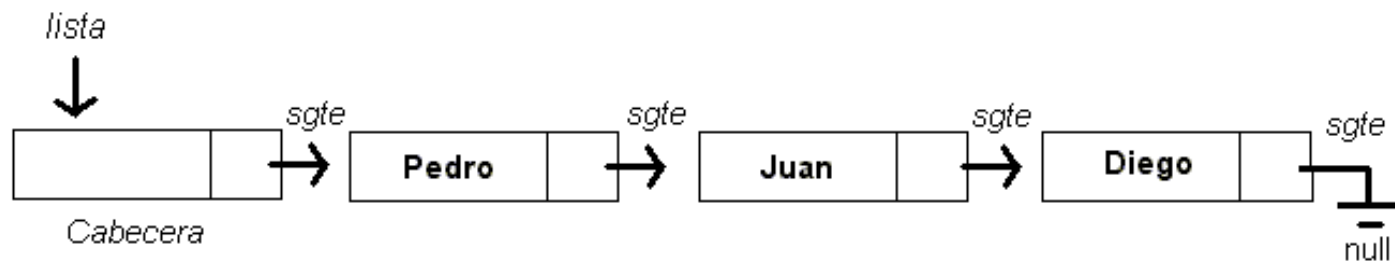
# Tipos de listas

- De acuerdo a su comportamiento, los **conjuntos lineales** se clasifican en
  - Listas, Pilas y Colas
- De acuerdo a su implementación, las **listas** se clasifican en:
  - Simplemente ligada o enlazada**
  - Doblemente ligadas o enlazadas**
    - Circulares



# Listas simplemente ligadas

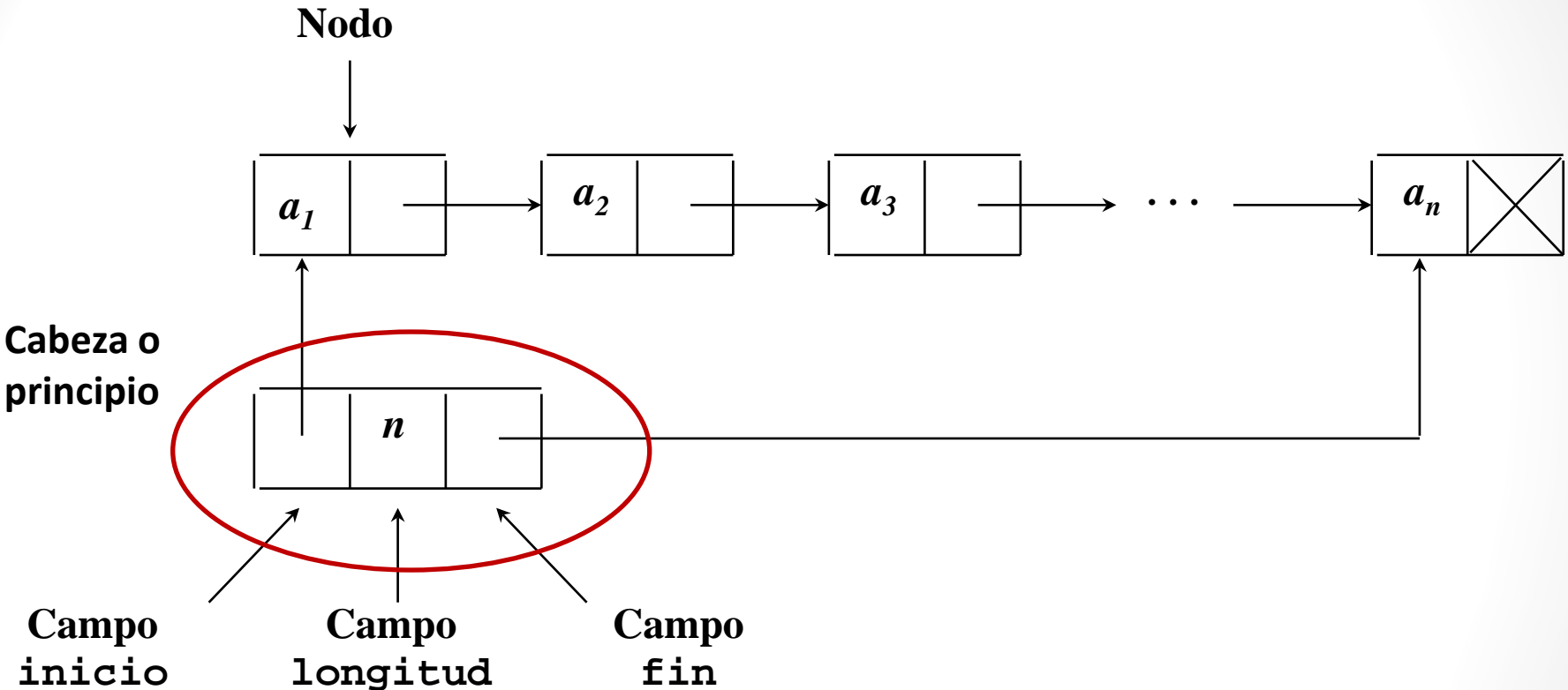
- Se define como un conjunto de nodos
  - Uno detrás de otro
  - Del cual siempre se puede conocer al nodo **inicial** y al **final**
- De cada nodo de la lista, se conoce
  - Un contenido, que es la información que almacena dentro
    - Puede ser de cualquier tipo de dato
  - Un sucesor único
    - Excepto el ultimo nodo de la lista







# Listas simplemente ligadas





# Listas simplemente ligadas!= Arreglo

- Arreglos tienen una longitud fija
- Para expandirlos es necesario crear un nuevo arreglo (de longitud mayor), y copiar el contenido del viejo arreglo al nuevo.
  - ¡Muy lento! ¡La peor opción!
- Una mejor solución es usar apuntadores entre los elementos del arreglo.
  - Cada elemento apunta al elemento siguiente.
  - Esto es una lista simple (*lista ligada*).





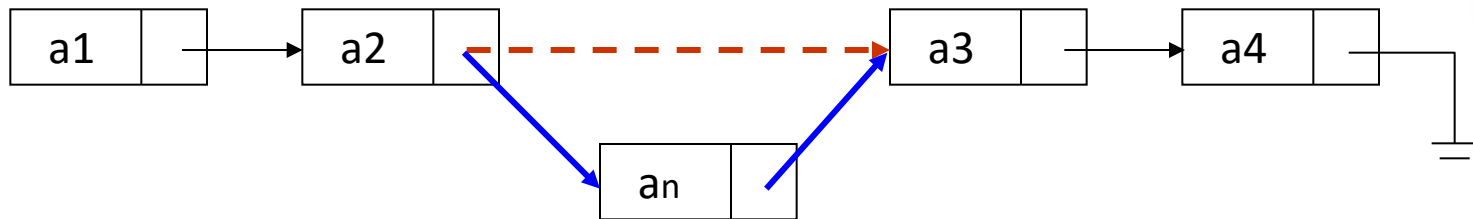
# Listas simplemente ligadas

- El modelo conceptual es como una “cadena”
  - Nuevos eslabones pueden insertarse al principio o al final, e incluso en cualquier otra posición con algo mas de esfuerzo.
  - Las extracciones pueden hacerse simplemente rompiendo la cadena y uniéndola nuevamente.
- Gran ventaja: ¡es dinámica!
  - **Se usa únicamente el espacio necesario.**
  - No es necesario conocer de antemano que tan grande será la estructura.

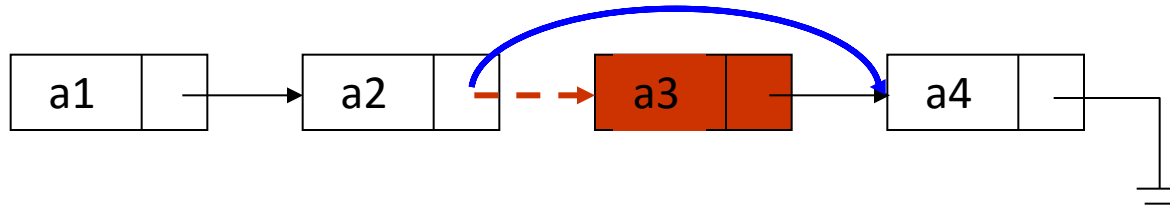


# Operaciones básicas de una lista simplemente ligada

- Insertar un nuevo elemento a la lista:

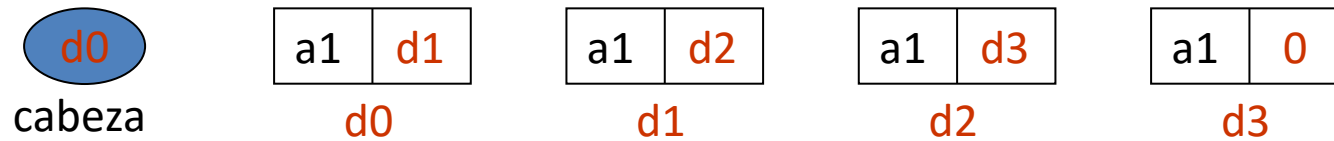
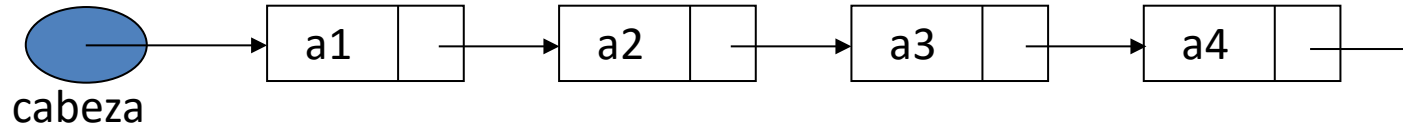


- Eliminar:





# Ejemplo de lista simplemente ligada



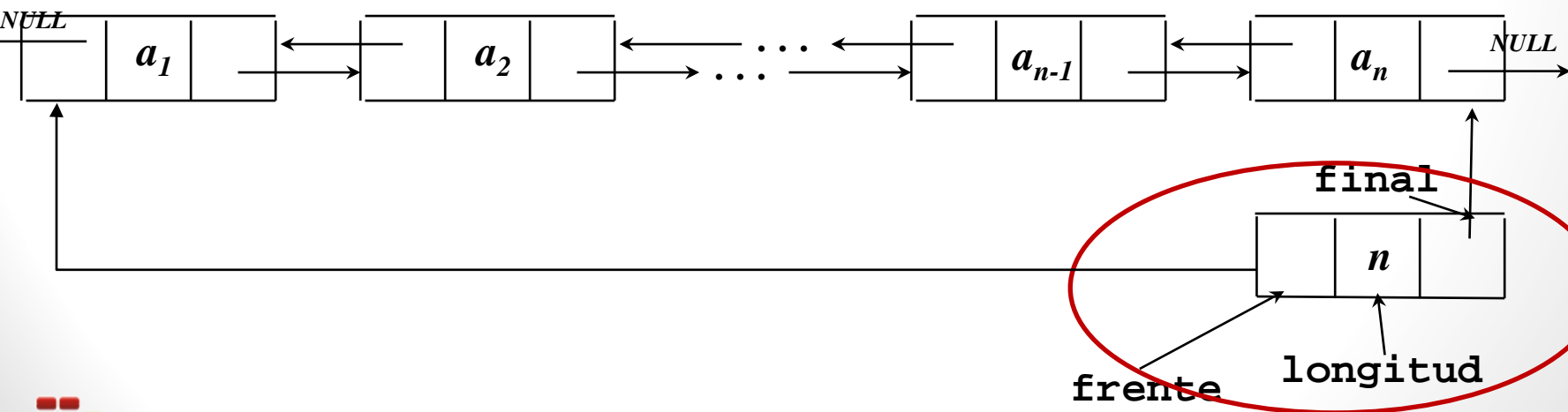
- Los elementos de la lista NO necesariamente están en localidades de memoria adyacentes.
- Para tener acceso a la lista únicamente es necesario conocer la localidad de memoria donde comienza (**d0**)





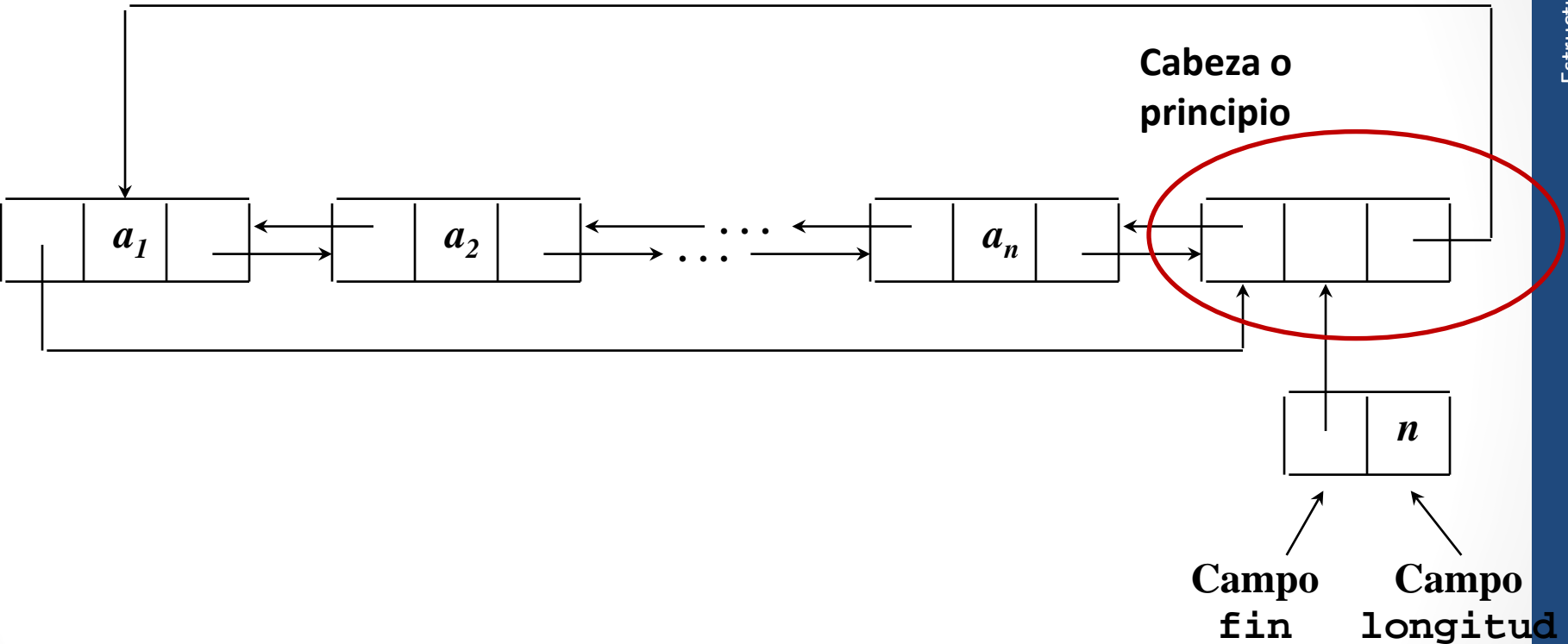
# Listas doblemente ligadas

- Una lista doblemente enlazada tiene punteros conectando los nodos en ambas direcciones. Esto permite recorrer la lista en ambas direcciones.



# Listas doblemente ligadas

Doble enlace y circular





# Aplicaciones del TAD Lista

- Las listas enlazadas son usadas como módulos para otras muchas estructuras de datos, tales como pilas, colas y sus variaciones.
- El campo de datos de un nodo puede ser otra lista enlazada. Mediante este mecanismo, se pueden construir muchas estructuras de datos enlazadas con listas; esta practica tiene su origen en el lenguaje de programación Lisp, donde las listas enlazadas son una estructura de datos primaria (aunque no la única), y ahora es una característica común en el estilo de programación funcional.







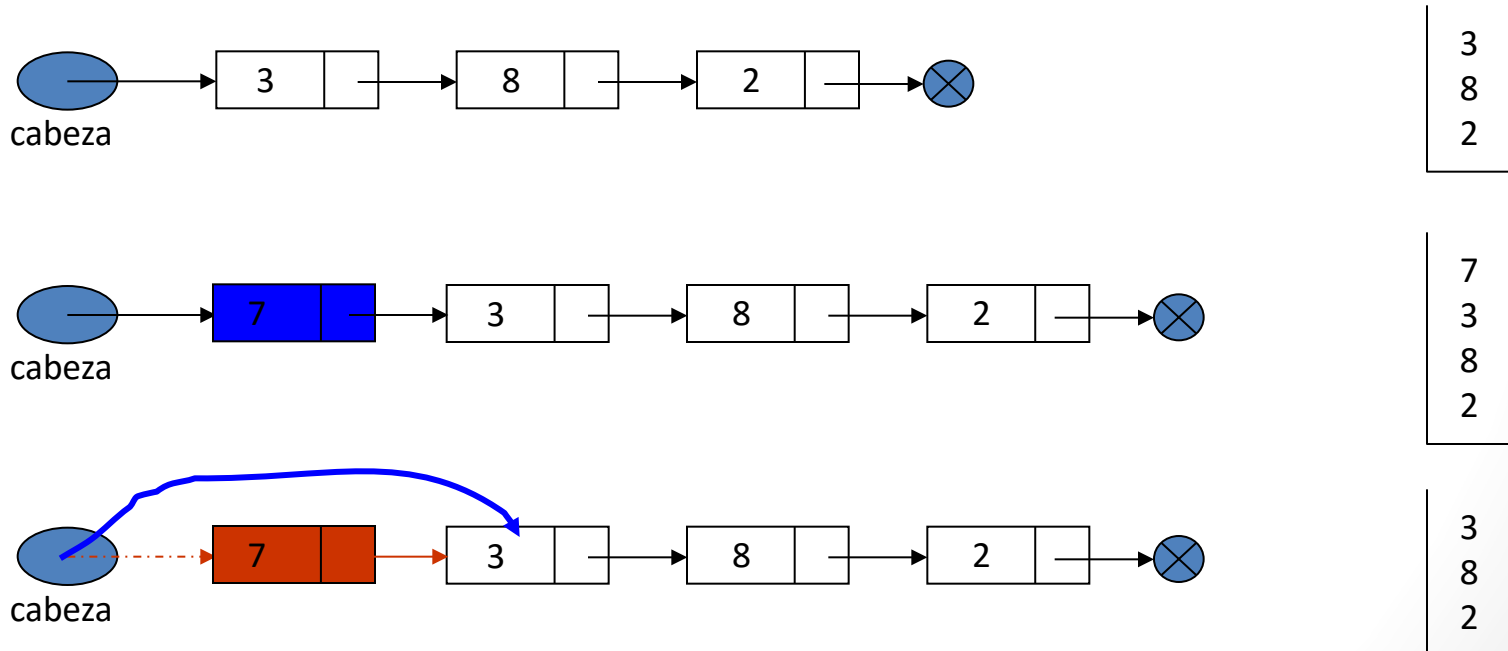
- A veces, las listas enlazadas son usadas para implementar vectores asociativos, y estas en el contexto de las llamadas listas asociativas.
- Hay pocas ventajas en este uso de las listas enlazadas; hay mejores formas de implementar éstas estructuras, por ejemplo con árboles binarios de búsqueda equilibrados. Sin embargo, a veces una lista enlazada es dinámicamente creada fuera de un subconjunto propio de nodos semejante a un árbol, y son usadas más eficientemente para recorrer ésta serie de datos.



# Pilas como listas

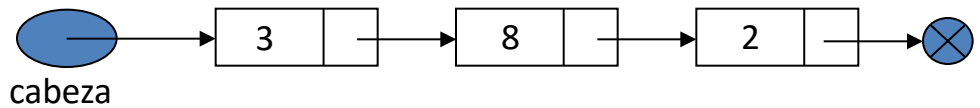
- **Idea básica:**

- Siempre se inserta y elimina del frente de la lista
- ¡Muy eficiente! No hay necesidad de recorrer la lista.

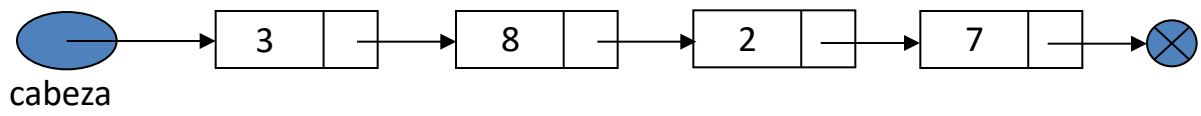


# Colas como listas

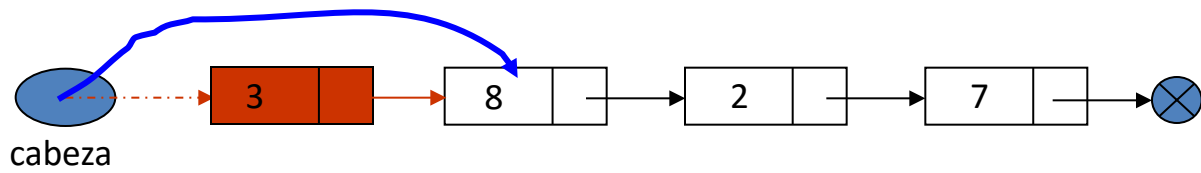
- **Idea básica:**
  - Se inserta al final, se elimina del frente.
  - Para evitar recorrer la lista en cada inserción se puede usar un apuntador al último elemento.



3 8 2



3 8 2 7



8 2 7





# Otras modalidades de las listas

- Además de las pilas o colas podemos modelar:
  - **Dicolas** (*Las eliminaciones e inserciones pueden realizarse en ambos extremos de la lista*).
  - **Colas de prioridad** (*A los elementos se les ha asignado una prioridad, de forma que el orden en que los elementos son desencolados respeta dicha prioridad*).
  - **Listas ordenadas** (*Los elementos dentro de la lista se encuentran en algún orden según sus características*).
  - **Listas circulares** (*Como en una lista enlazada simple, los nuevos nodos pueden ser solo eficientemente insertados después de uno que ya tengamos referenciado debido a la circularidad de la lista*).

