DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Estructuras de Datos

Pamela Landero

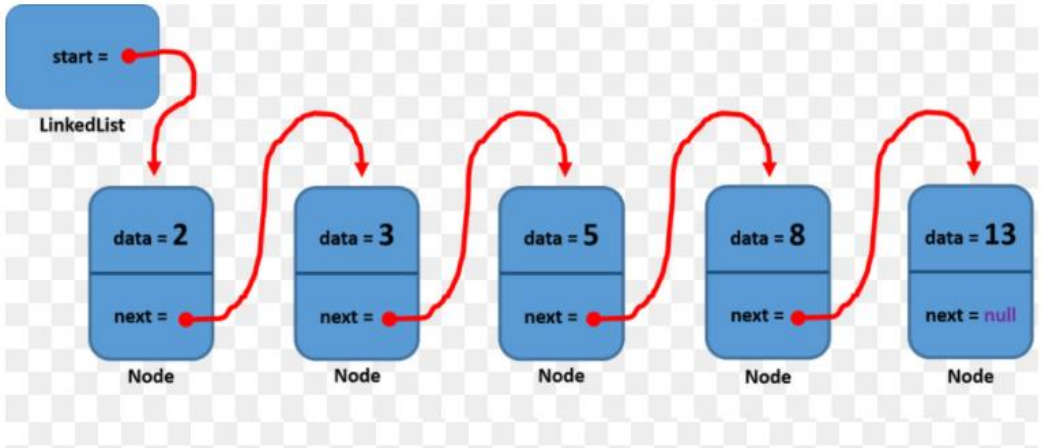
Clase COORDINADA semana 6 - Unidad 2

Semestre 2-2022

Contenido

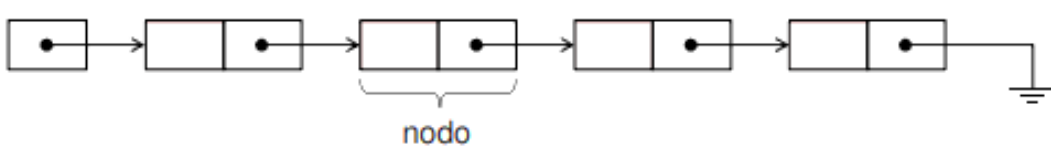
- TDA Lista simplemente enlazada (nodos)
- Otros tipos de listas o colas: circulares y doblemente enlazadas
- Aplicaciones
- Resumen

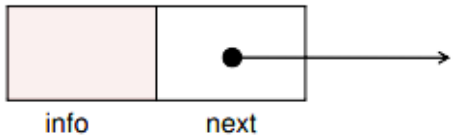
TDA Lista simplemente enlazada



TDA Lista: Representación Dinámica/ Lista enlazada

- **Lista enlazada:** es una colección o secuencia de elementos, llamados **nodos**, dispuestos uno detrás de otro, en la que cada elemento se conecta al siguiente por un enlace (puntero).





- Lista de nodos, donde nodo:
  - ✓ La primera parte contiene la ***información***.
  - ✓ Y la segunda es un campo de tipo puntero (***enlace*** o ***next***) que apunta al siguiente elemento de la lista.
- Esta representación permite un uso más eficiente de la memoria generando elementos en la lista sólo en la medida que se requiera.
- Las modificaciones se realizan en los elementos de la lista que corresponden y no afectan a los otros elementos de la lista.

TDA Lista : Representación Dinámica/Lista enlazada

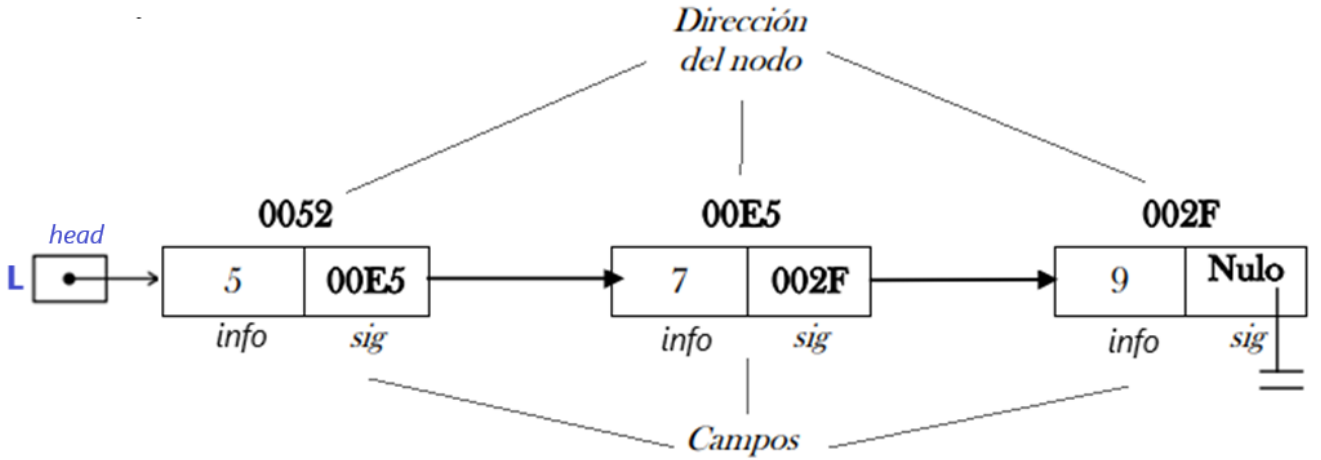
- *Los nodos se crean/destruyen dinámicamente:*
  - ✓ Uso de memoria dinámica
  - ✓ Creación de nodos => malloc
  - ✓ Liberación de nodos => free
- *Ventajas*:
  - ✓ Sólo se tiene reservada la memoria que se necesita en cada momento.
  - ✓ Se pueden albergar tantos elementos como la memoria disponible permita.
  - ✓ Insertar/eliminar nodos no requiere desplazamientos de memoria.
- *Desventajas*:
  - ✓ Es bueno para el acceso secuencial – es malo para el acceso aleatorio
- *Clasificación de las listas enlazadas:*
  - ✓ ***Listas simplemente enlazadas***
  - ✓ Listas doblemente enlazadas
  - ✓ Listas circular simplemente enlazadas
  - ✓ Listas circular doblemente enlazadas

TDA Lista: Representación de su estructura de datos

✓ puntero *head* (inicio) de la lista

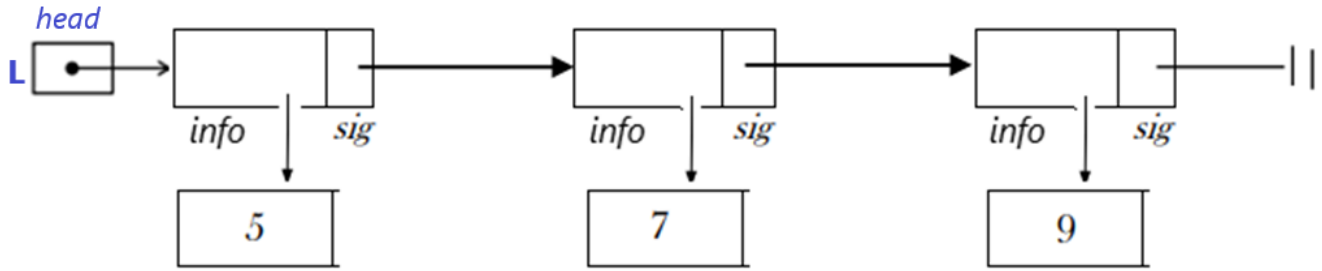
REPRESENTACIÓN 1:
(datos en el mismo struct nodo)

Nodo:
info: tipo_dato
sig**: ***Nodo
Lista:
head**: ***Nodo
n: Entero



REPRESENTACIÓN 2:
(datos en struct Info)
(CASO GENERAL)

Info:
dato: tipo_dato
Nodo:
info**: ***Info
sig**: ***Nodo
Lista:
head**: ***Nodo
n: Entero



TDA Lista Dinámica- Operaciones

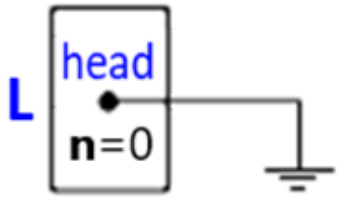
crearLista()	⇒ Lista	Crea y retorna una lista vacía (sin nodos). Inicializa los datos definidos en la estructura Lista dependiendo de su representación y retorna la dirección de memoria de la lista creada.
crearNodo()	⇒ Nodo	Crea en memoria un nodo, asigna sus datos y retorna la dirección de memoria del nodo creado. Si la data del nodo está en otra estructura (según representación) debe considerar además crear la data con su respectiva asignación de memoria.
insertarNodoIni(L,x)	⇒ Lista	Crea un nodo y lo Inserta al inicio de la lista con elemento x al inicio de la lista L (donde está el head).
insertarNodoFin(L,x)	⇒ Lista	Inserta nodo con elemento x al final de la lista L. La función recorre la lista hasta llegar al último nodo y después lo inserta.
***insertarNodo*(L,x, z)**	⇒ Lista	Inserta nodo con elemento x después de un elemento dado z de la lista L. Si z es repetido, inserta después del primero.
***recorrerLista*(L)**	⇒ void	Recorre la lista para visualizar la información que contiene cada nodo de la lista.
buscarNodo(L,x)	⇒ Nodo	Entrega el nodo (dirección de memoria) donde se encuentra x en L. Si x está repetido, entrega la primera. El resultado queda indefinido si L es vacía.
primero(L)	⇒ Nodo	Entrega el nodo (dirección de memoria) que se encuentra al inicio de L. El resultado queda indefinido si L es vacía.
ultimo(L)	⇒ Nodo	Entrega el nodo (dirección de memoria) que se encuentra al final de L. El resultado queda indefinido si L está vacía.
eliminarNodoIni(L)	⇒ elem	Elimina el nodo que se encuentra al inicio de la lista L (donde está el head). El resultado queda indefinido si L es vacía.
eliminarNodoFin (L)	⇒ elem	Elimina el nodo que se encuentra al final de la lista L.. El resultado queda indefinido si L es vacía.
eliminarNodo(L,x)	⇒ void	Elimina nodo con elemento x de la lista L. Si x está repetido, elimina el primero que encuentra. El resultado queda indefinido si L es vacía. Si el nodo se encuentra al inicio o al final ejecutar operaciones correspondientes.
actualizarLista(L, x,y)	⇒ Lista	Actualiza el nodo con elemento x con y. El resultado queda indefinido si L no tiene x.
largoLista(L)	⇒ n	Entrega el número de nodos que posee L.
isListaVacía(L)	⇒ V, F	Indica si L está vacía (V).

TDA Lista: algoritmo *crear lista*

```
Lista:
    *head: Nodo
    n:Entero

Nodo:
    info: tipo_dato
    *sig: Nodo
```

```
Lista *crearLista( ):
Inicio
   *L: Lista
   L: Asignar memoria
   L->head=Nulo
   L->n = 0
   retornar L
Fin
```



TDA Lista: algoritmo *Insertar al principio*

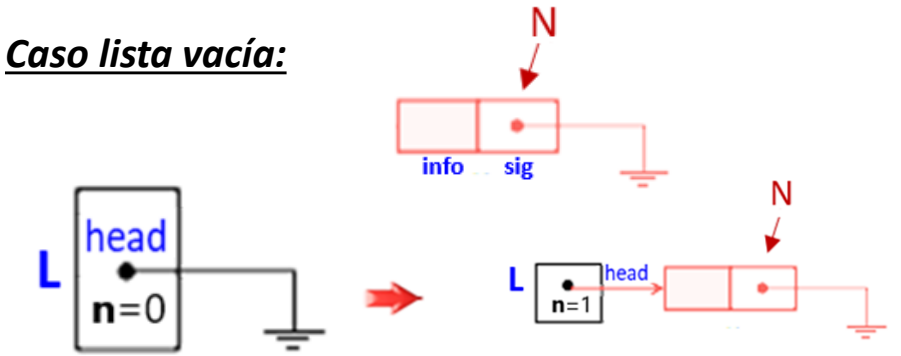
Lista:
*head: **Nodo**
n:Entero

Nodo:
info: tipo_dato
sig**: **Nodo

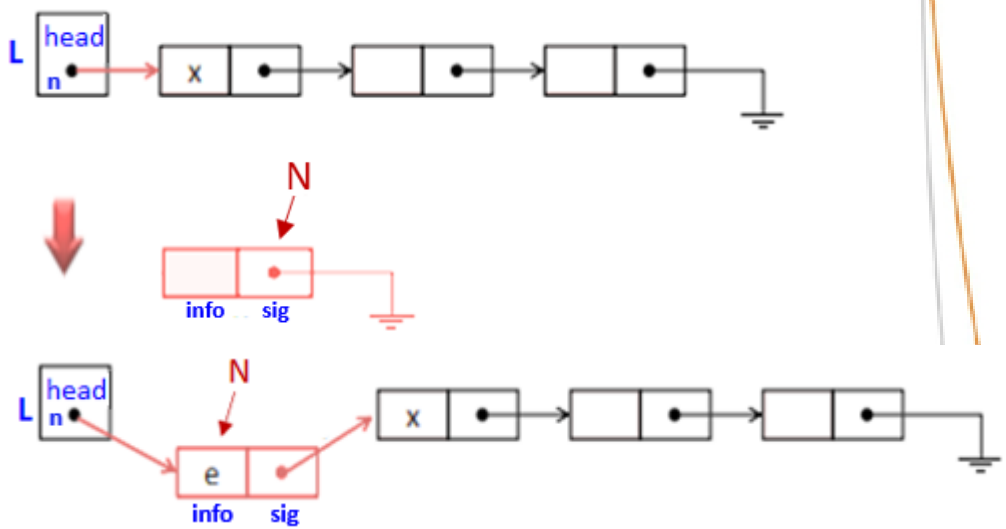
```
insertarNodoIni (*L: Lista, val: tipo_dato)
Inicio
   *N: Nodo
   N=crearNodo(val)
   Si  not(isListaVacía(L)) entonces
         N->sig = L->head
   FinSi
    L->head = N
    L->n = L->n  + 1
Fin
```

```
Nodo crearNodo(val: tipo_dato)
Inicio
   *nuevoNodo: Nodo
   nuevoNodo: Asignar memoria
   nuevoNodo->info = val
   nuevoNodo->sig = Nulo
   retornar nuevoNodo
Fin
```

Caso lista vacía:



Caso lista NO vacía:

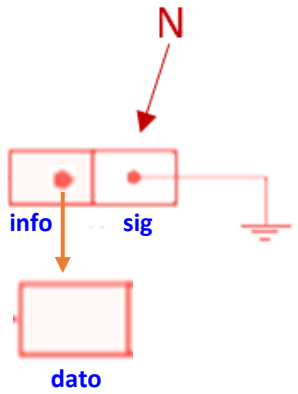


TDA Lista: algoritmo *Insertar al principio*

Lista:
*head: **Nodo**
n:Entero

Nodo:
*info: **Info**
*sig: **Nodo**

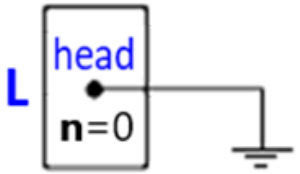
Info:
dato: tipo_dato



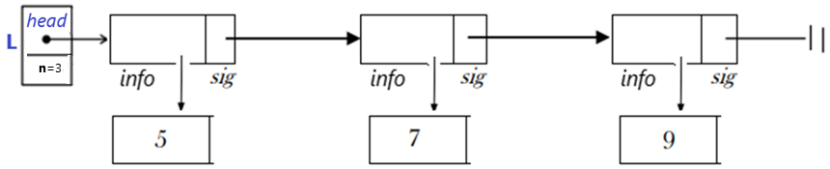
insertarNodoIni ( )
Inicio

Fin

Caso lista vacía:



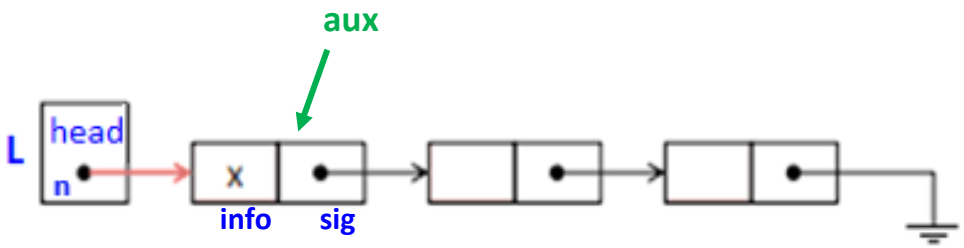
Caso lista NO vacía:



TDA Lista: Algoritmo *recorrer lista*

Lista:
head*: ***Nodo*
n:*Entero*

Nodo:
info: tipo_dato
sig**: ***Nodo




```
recorrer_lista(*L: Lista )
Inicio
   *aux: Nodo
   aux =  L->head
   Mientras (aux ≠ Nulo)
      Escribir(aux->info)
      aux = aux->sig
   FinMientras
Fin
```

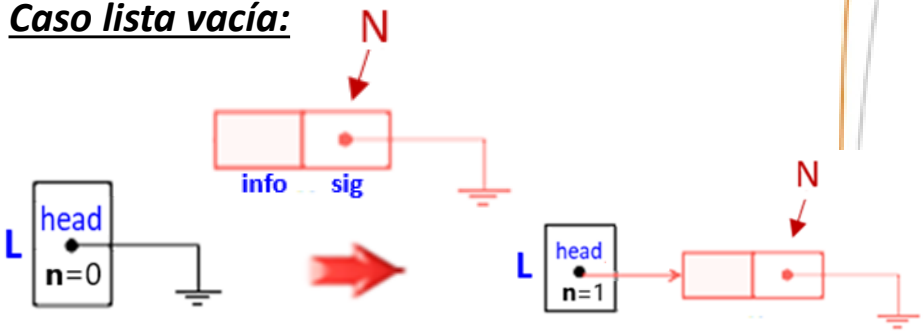
TDA Lista: Algoritmo *Insertar al final*

```
Lista:
    *head: Nodo
    n:Entero
Nodo:
    info: tipo_dato
    *sig: Nodo
```

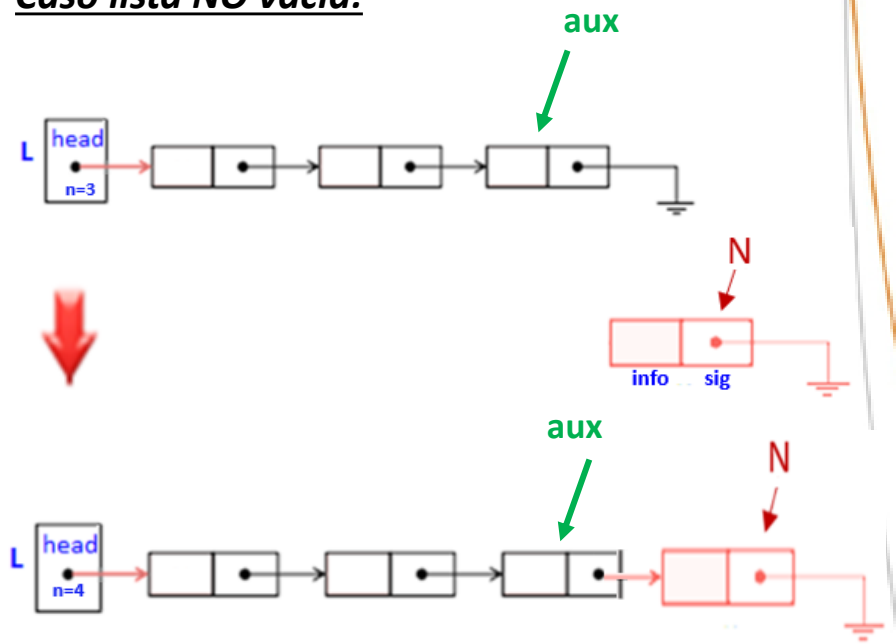
```
Nodo *último(*L: Lista)
Inicio
    *a: Nodo
    a = L->head
    Mientras (a->sig ≠ Nulo)
        a = a->sig
    FinMientras
    Regresar a
Fin
```

```
insertarNodoFin (*L: Lista, val: tipo_dato)
Inicio
    *N: Nodo N
    N=crearNodo(val)
    Si (isListaVacía(L)) entonces
        L->head = N
    Sino
        *aux: Nodo
        aux = último( L)
        aux-> sig = N
    FinSi
    L->n = L->n  + 1
Fin
```

Caso lista vacía:



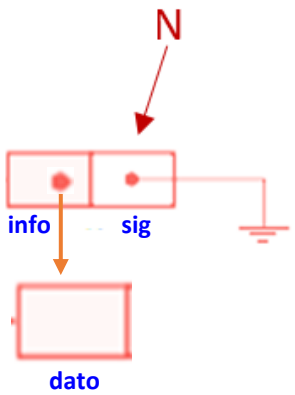
Caso lista NO vacía:



TDA Lista: Algoritmo *Insertar al final*

Lista:
*head: **Nodo**
n:Entero
Nodo:
*info: **Info**
*sig: **Nodo**

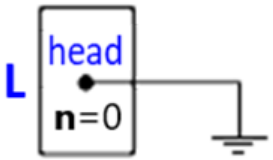
Info:
dato: tipo_dato



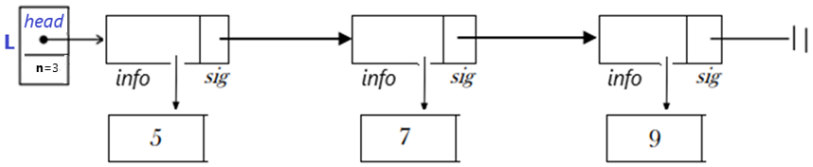
insertarNodoIni ( )
Inicio

Fin

Caso lista vacía:



Caso lista NO vacía:

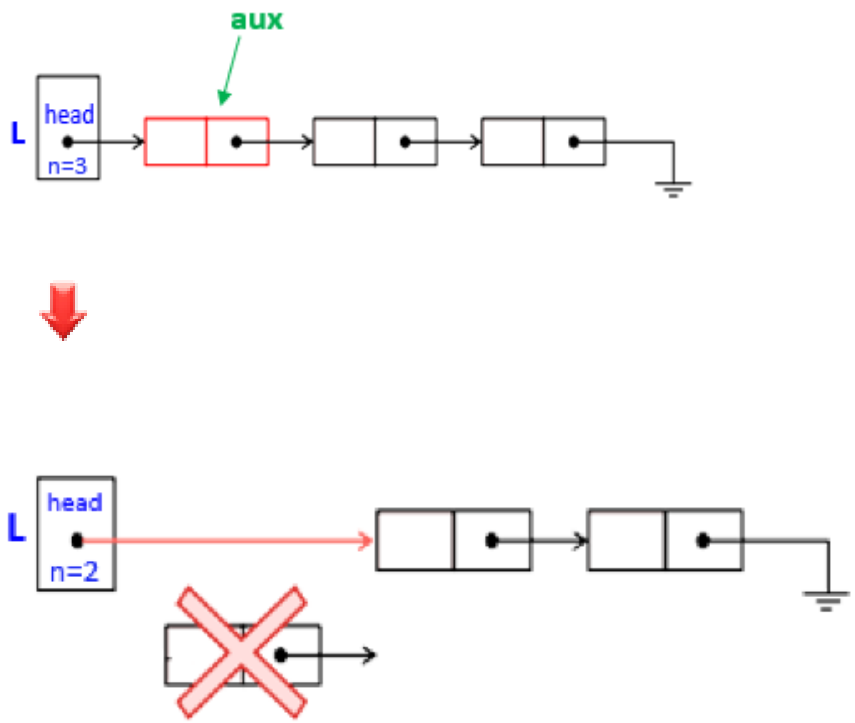


TDA Lista: algoritmo *eliminar al inicio*

Lista:
*head: **Nodo**
n:Entero

Nodo:
info: tipo_dato
sig**: **Nodo

```
tipo_dato  eliminarIni(*L: Lista)
Inicio
  Si not(isListaVacía(L)) entonces
     *aux: Nodo
     aux =  L->head
     L->head = L->head->sig
     aux->sig = Nulo
     e = aux->info
     Liberar (aux)
     L->n = L->n  - 1
     Regresar e
  FinSi
Fin
```



TDA Lista: algoritmo *Eliminar al final*

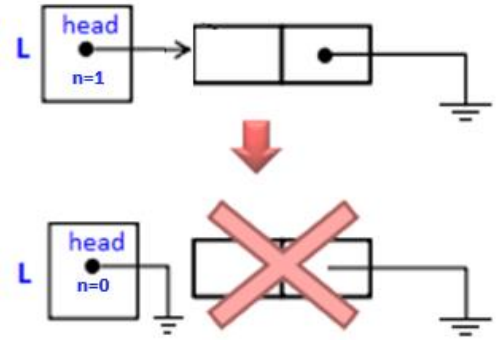
```
Lista:
    *head: Nodo
    n:Entero

Nodo:
    info: tipo_dato
    *sig: Nodo
```

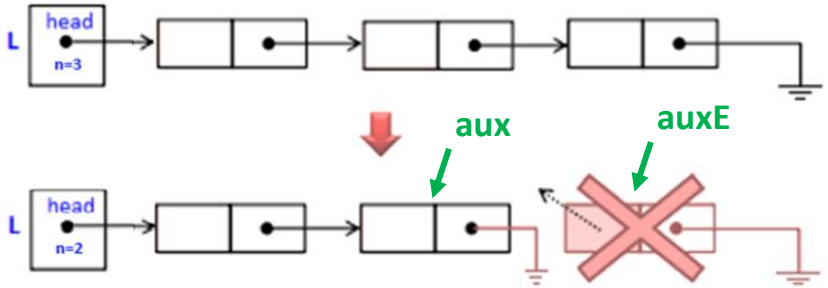
```
Nodo *penúltimo(*L: Lista)
Inicio
    *a: Nodo
     a = L->head
    Mientras (a->sig->sig ≠ Nulo)
        a = a->sig
    FinMientras
    Regresar a
Fin
```

```
tipo_dato eliminarFin(*L: Lista)
Inicio
  Si not(isListaVacía(L)) Entonces
    *aux,*auxE: Nodo;
     aux = penúltimo( L)
     auxE = aux->sig
     aux->sig = Nulo
     e = auxE->info
     Liberar (auxE)
     L->n = L->n  - 1
    Regresar e
  FinSi
Fin
```

Caso lista con número nodos = 1:



Caso lista cantidad nodos > 1 :



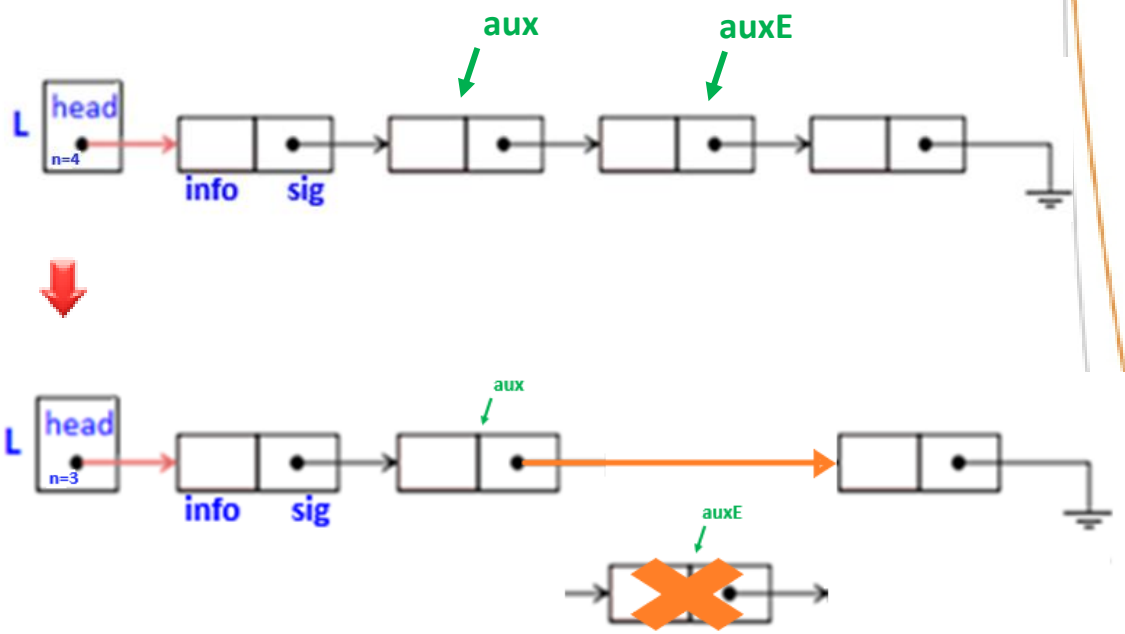
TDA Lista: algoritmo *Eliminar nodo con valor dado*

```
Lista: Registro
        *head: Nodo
        n:Entero
Nodo:
        info: tipo_dato
        *sig: Nodo
```

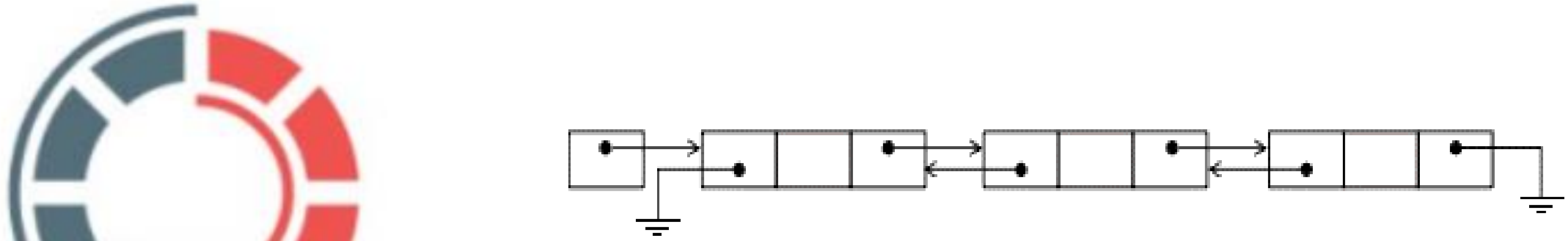
```
Nodo buscar (val:E, *L:Lista)
Inicio
    *aux: Nodo
    aux = L->head
    Mientras(aux ≠ nulo)
       Si (aux->info ≠ valor) entonces
              aux = aux->sig
       Sino
              Regresa aux  /* lo encuentra */
       FinSi
    FinMientras
    Regresa nulo /* no lo encuentra */
Fin
```

```
eliminar (*L:Lista, val:tipo_dato)
Inicio
    *auxE: Nodo
    Si not(isListaVacía(L)) Entonces
        auxE = buscar(val,L)
        Si (auxE ≠ Nulo) entonces
                Si (auxE = L->head) entonces
                  e = eliminarIni(L)
                Sino
                  Si (auxE->sig = Nulo) entonces
                        e = eliminarFin(L)
                  Sino
                        aux = anterior(L,auxE);
                        aux->sig = auxE>sig ;
                        Liberar (auxE);
                        L->n = L->n  - 1
                  FinSi
                FinSi
        FinSi
    FinSi
Fin
```

```
Nodo * anterior(Lista *L, *N:Nodo)
Inicio
      *aux: Nodo
      aux = L->head
      Mientras (aux->sig ≠ N)
            aux = aux->sig
      FinMientras
      Regresa aux
Fin
```

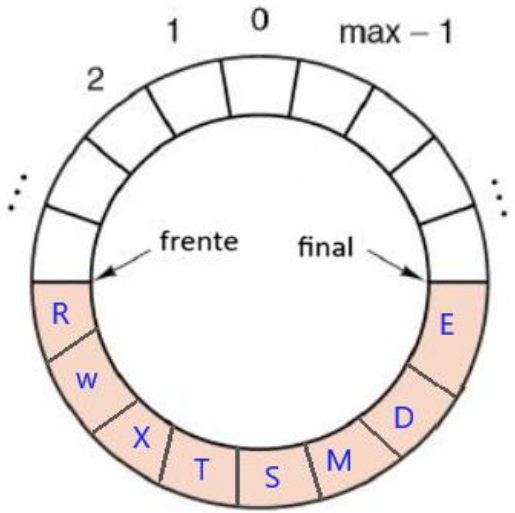


Otros tipos de listas/colas:
circular - doblemente enlazada



TDA Cola circular estática:

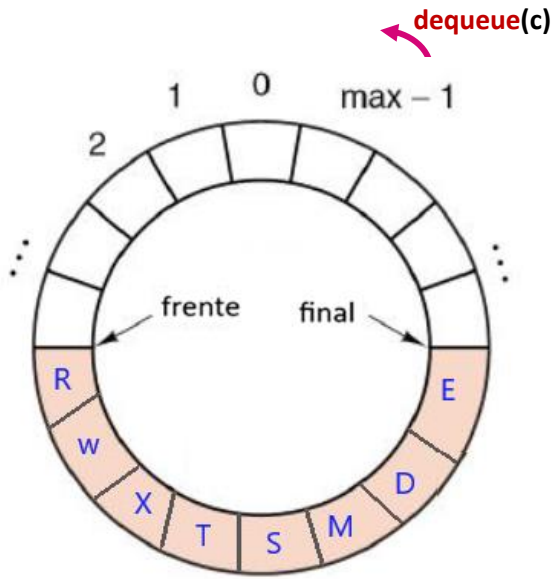
- Cuando se realiza la implementación estática (con arreglos unidimensionales) de una **cola** (o ***queue***), los espacios que van quedando en las celdas al eliminar los elementos de ésta, se "desperdician". Esto se aprecia en figura del lado izquierdo donde las celdas, desde la posición ***0*** a la posición ***frente- 1,*** quedan vacías y no es posible reutilizarlas aunque frente llegue a ***max-1*** (cola full).
- En una cola circular (arreglo circular) se considera que la primera posición sigue a la última, es decir, el elemento anterior al primero es el último, o bien, al último elemento le sigue el primero. Esto implica que aún ocupado el último elemento del arreglo, pueda añadirse uno nuevo detrás de éste, ocupando la primera posición del arreglo.
- La diferencia de implementación de las operaciones, comparada con la versión no circular, existe en ***determinar correctamente cuando la cola está llena y cuándo está vacía***; en la operación de ***encolar*** (insertar) añadir uno nuevo en la primera posición, si es posible, cuando final esté en ***max-1;*** y en la operación ***desencolar*** (eliminar) extraer el elemento siguiente al último de la cola, si es posible.



TDA Cola circular: Operaciones (algoritmos)

OPERACIÓN	ALGORITMO	PSEUDOCÓDIGO
isEmpty (Cola vacía)	1. Verificar si el frente de la cola es igual al final de la cola. Si son iguales regresa un true, en caso contrario regresa un falso.	**Boolean isEmpty**(*c: Cola) ***Inicio*** Si (c->frente and c->final = -1)entonces regresa ***True*** Sino *regresa* ***False*** FinSi ***Fin***
isFull (Cola LLena)	1. Verificar si el final de la cola (MAX - 1) es mayor que el número máximo de elementos permitidos en el arreglo especificado en la declaración de la cola.	Boolean ***isFull***(*c: Cola) ***Inicio*** Si (c->final = MAX-1 y c->frente = 0) **o** (c->(final+1) = c->frente) entonces regresa ***True*** Sino regresa ***False*** FinSi ***Fin***
front (recuperar elemento del frente)	1. Verificar si la cola no está vacía. 2. Recuperar el elemento situado en la posición especificado por el puntero frente de la cola en el arreglo.	***front***(*c: Cola) ***Inicio*** Si (***isEmpty***(c)) entonces Escribir (“Cola Vacía") Sino **regresa** c->datos[c->frente] ***Fin***

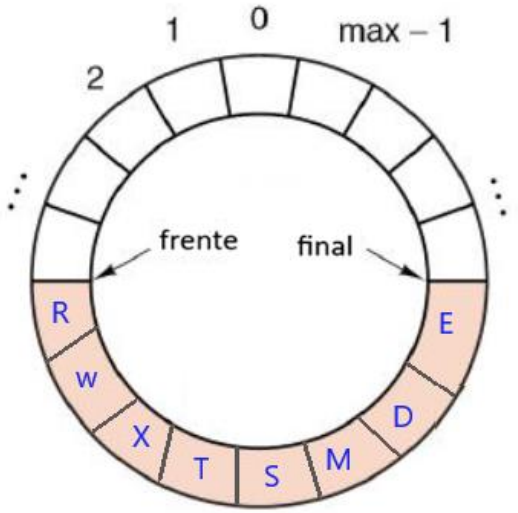
Cola:
datos[MAX]:Entero
frente: ***Entero***
final: ***Entero***
:
**c: Cola*



TDA Cola circular: Operaciones (algoritmos)

OPERACIÓN	ALGORITMO	PSEUDOCÓDIGO
enqueue (Insertar)	1. Verificar si la cola no está llena. ***2. Si no está llena la cola hacer:*** ✓ Si el final es el máximo número de elementos (MAX-1), ***asignar 0*** al final de la cola, en caso contrario, incrementarlo en 1. ✓ Almacenar el elemento nuevo en la posición del puntero final de la cola. ✓ Si el puntero frente es 0 asignarle 1.	***enqueue***(*c: Cola, val: Entero) ***Inicio*** Si (***isFull***(c)) entonces Escribir ("Cola llena") Sino Si (c->final = MAX-1) entonces c->final = 0 sino c->final = c->final + 1 FinSi c->datos[c->final] = val Si (c->frente = -1) entonces //inserta C vacía c->frente = 0 FinSi FinSi ***Fin***

Cola:
datos[MAX]:Entero
frente: ***Entero***
final: ***Entero***
:
**c: Cola*

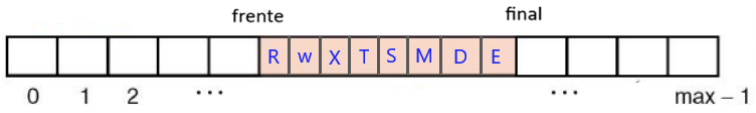


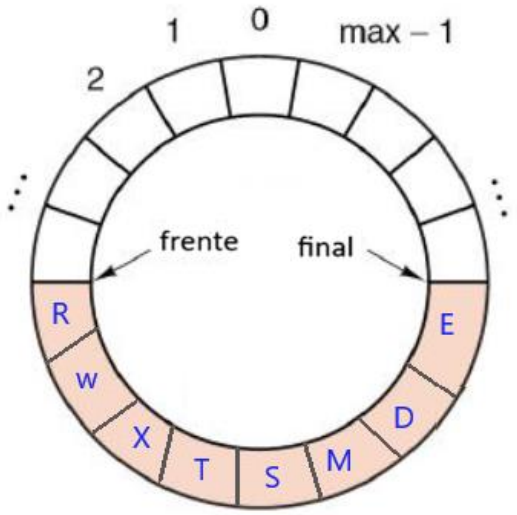
TDA Cola circular: Operaciones (algoritmos)

OPERACIÓN	ALGORITMO	PSEUDOCÓDIGO
dequeue (Quitar o extraer)	1. 1. Verificar si la cola no está vacía 2. 2. Verificar que frente y final no sean 3. iguales, si son iguales se les asigna 0 4. 3. Si el frente es igual al máximo de 5. elementos, a frente se le asigna 1, si 6. no es igual se incrementa en 1.	Entero ***dequeue***((*c: Cola) ***Inicio*** Si (***isEmpty***(c)) entonces Escribir (“Cola Vacía") Sino e = ***front***(c) Si (c->frente = c->final) entonces //reiniciar C c->frente = -1 c->final = -1 Sino Si (c->frente = MAX-1) entonces c->frente = 0 Sino c->frente = c->frente + 1 FinSi FinSi FinSi **regresa e** ***Fin***

Cola:
datos[MAX]:Entero
frente: ***Entero***
final: ***Entero***

**c: Cola*

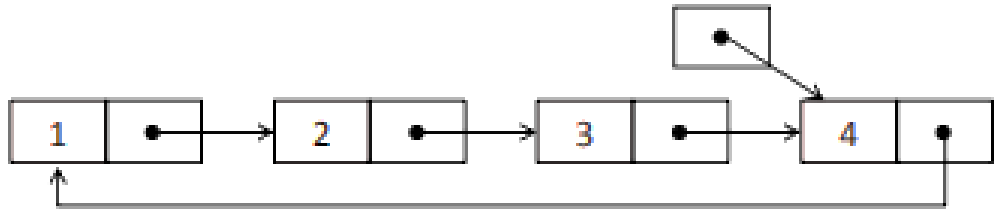




TDA **Lista enlazada** circular

¿Cómo resolver que al agregar/eliminar al final de una lista simplemente enlazada no sea costoso, pues hay que recorrer la lista para situarse en el (pen)último nodo?

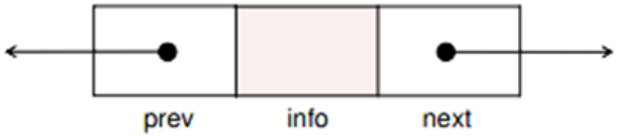
Una de las soluciones: Haciendo que el último nodo apunte al primero.

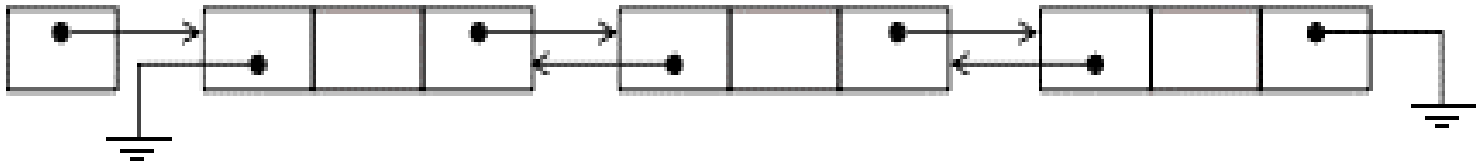


TDA Lista: Lista doblemente enlazada

¿Cómo resolver que al agregar/eliminar al final de una lista simplemente enlazada no sea costoso, pues hay que recorrer la lista para situarse en el (pen)último nodo?

Otra solución: doble puntero.





Resumen

- Estructuras de Datos.
- TDA Lista en su representación dinámica
- Clasificación Listas/Colas:
  - Simplemente enlazada
  - Doblemente enlazada
  - Simplemente enlazada circular
  - Doblemente enlazada circular
- TDA Colas y TDA Pila son situaciones particulares de Listas

Tarea:

- *Implementar en C el TDA Lista Estática.*
- *Implementar en C el TDA Lista Dinámica simplemente enlazada y comprenda las diferencias con sus otras clasificaciones*

Consultas