Tema Final: LISTAS - UNIDAD 3

INTRODUCCIÓN

**¿Que es una lista enlazada?**

Una lista enlazada es una estructura de datos dinámica formada por una colección de nodos conectados entre sí mediante enlaces. Cada nodo almacena dos cosas:

1. Información o datos.
2. Un enlace (puntero) que apunta al siguiente nodo en la lista.

Esto permite que los nodos puedan estar distribuidos en diferentes posiciones de la memoria, conectándose entre sí gracias a los punteros.

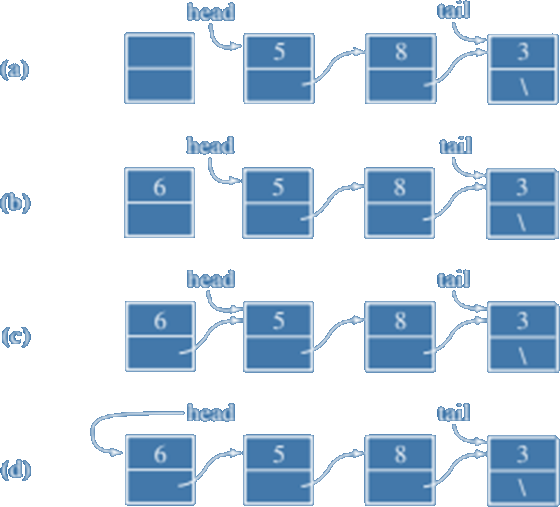
Las listas se pueden almacenar de manera dinámica, es decir, pueden crecer o reducirse en tamaño a medida que se agregan o eliminan elementos.

**En esta presentación mencionaremos cómo funcionan las operaciones básicas de las listas enlazadas simples y las definiciones de sus tipos como (**las listas doblemente enlazadas, las listas circulares y doblemente enlazadas circulares**).**

**—-------------------------------------------------------------------------------------------**

**INSERCIÓN AL INICIO DE LA LISTA:**

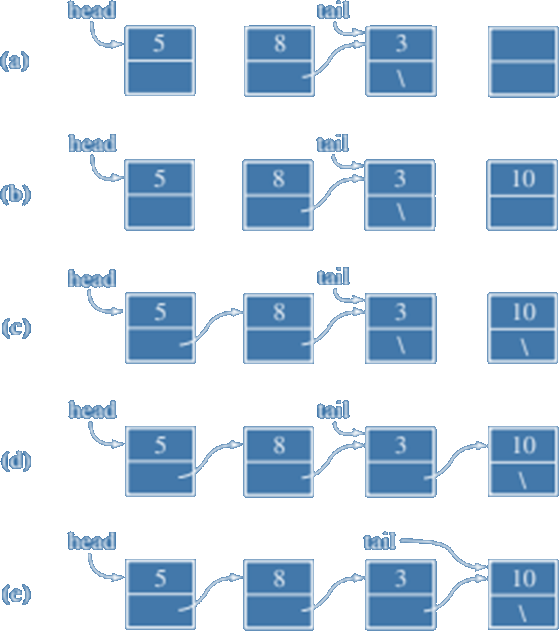
1. Al principio se crea un nodo vacío el cual no se asigna ningún valor a los miembros de datos del nodo (Figura a).

2. El miembro info del nodo se inicializa en un entero. (Figura b).

3. Debido a que el nodo se incluye al frente de la lista, el siguiente miembro se convierte en un puntero al primer nodo de la lista; es decir, el valor actual de la cabeza. (Figura c).

4. El nuevo nodo precede a todos los nodos de la lista, actualizando el cabezal ya que este puntero es el que indica el inicio de la lista. De lo contrario, no se podrá acceder al nuevo nodo. (Figura d).

**—-------------------------------------------------------------------------------------------**

**INSERCIÓN AL FINAL DE LA LISTA:**

1. Se crea un nodo vacío (Figura a).

2. El miembro info del nodo se inicializa en un entero (Figura b).

3. Debido a que el nodo se agrega al final de la lista, el siguiente miembro se establece en null para indicar que es el último (Figura c).

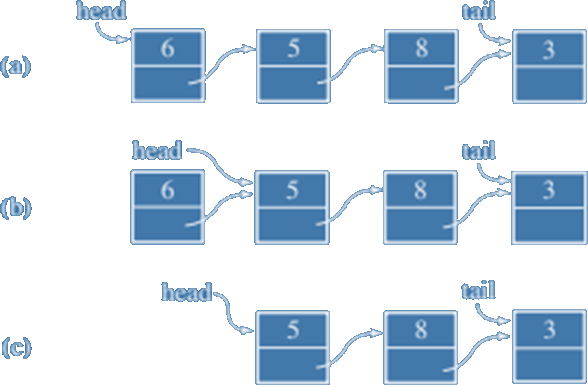
4. El nodo ahora se incluye en la lista haciendo que el siguiente miembro del último nodo de la lista apunte al último nodo recién creado (Figura d).

5. El nuevo nodo sigue a todos los nodos de la lista, actualizando el valor de “tail” para señalar al último nodo de la lista. (Figura e).

**—------------------------------------------------------------------------------------**

**ELIMINAR EL NODO AL PRINCIPIO DE LA LISTA (DELETE HEAD)**

Consiste en eliminar un nodo al principio de la lista y devolver el valor almacenado en él.

En esta operación, la información del primer nodo se almacena temporalmente en una variable local , y luego se restablece el cabezal para que lo que era el segundo nodo se convierta en el primer nodo.

También hay dos casos especiales a considerar.

1. Cuando intentamos eliminar un nodo en una lista vacía, hay que hacer una excepción.
2. Por lo que hay que declarar afirmaciones para que devuelva un número o algún dato.

**int\* IntSLList::deleteFromHead() {**

**if (!isEmpty()) {**

**int \*el = new int(head->info);**

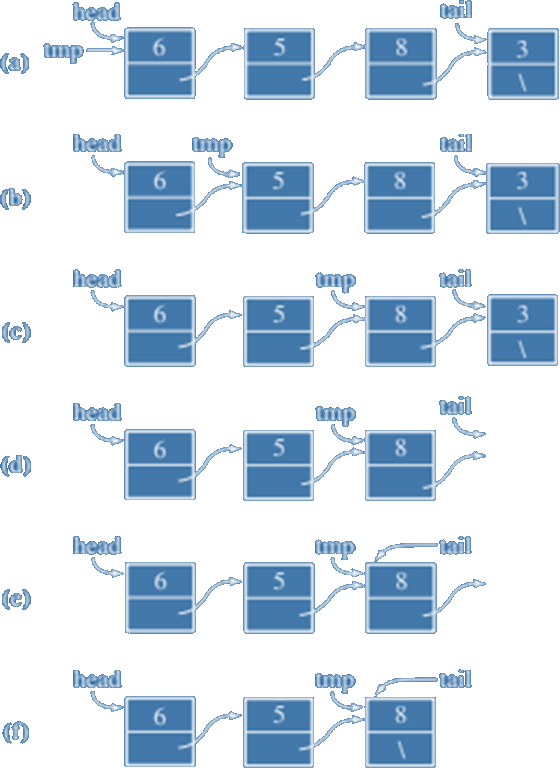
**return el;**

**}**

**else return 0;**

**}**

**—------------------------------------------------------------------------------------**

**ELIMINAR EL ÚLTIMO NODO DE LA LISTA (DELETE END)**

Después de eliminar un nodo, la cola (`tail`) debe actualizarse para referirse al nuevo último nodo de la lista, lo que requiere mover la cola un nodo hacia atrás.

Para lograrlo, se utiliza una variable temporal (`tmp`) que recorre la lista desde el inicio. La variable `tmp` se inicializa apuntando al nodo de la cabeza (`head`) y avanza nodo por nodo dentro de un bucle, deteniéndose en el penúltimo nodo (es decir, el nodo anterior al último).

Cuando se encuentra el penúltimo nodo, el nodo final se elimina y la cola (`tail`) se actualiza para apuntar al penúltimo nodo, que ahora se convierte en el último de la lista. Finalmente, para marcar este cambio, el puntero `siguiente` del nuevo nodo final de la lista se establece en `null`.

**—------------------------------------------------------------------------------------**

**Eliminar en cualquier lugar dentro de una lista enlazada (any where)**

Consiste en localizar el nodo que se desea eliminar y ajustar los enlaces de los nodos adyacentes para que la lista mantenga su estructura. Hay varios casos para borrar el nodo según su posición:

* Un intento de eliminar un nodo de una lista vacía, en cuyo caso la función se cierra inmediatamente.
* Eliminación del único nodo de una lista vinculada de un nodo: tanto la cabeza como la cola se establecen en null.
* Eliminar el primer nodo de la lista con al menos dos nodos, lo que requiere actualizar el encabezado.
* Eliminar el último nodo de la lista con al menos dos nodos, lo que lleva a la actualización de la cola.
* Un intento de eliminar un nodo con un número que no está en la lista: no hacer nada.

**—------------------------------------------------------------------------------------**

**BÚSQUEDA DE UN NODO EN LA LISTA (SEARCH)**

La operación de **búsqueda** se utiliza para verificar si un elemento (por ejemplo, un número) está presente en la lista. A continuación, explicación de cómo funciona:

Recorrido de la lista:

* Se utiliza un puntero temporal (tmp) que comienza apuntando al primer nodo de la lista (la cabeza o head).

Comparación:

* En cada nodo, se compara el dato almacenado en ese nodo con el número que estás buscando.

Resultado de la comparación:

* Si el número del nodo coincide con el número buscado, se detiene el recorrido y se devuelve true (el número está en la lista).
* Si no coincide, se avanza al siguiente nodo (tmp = tmp->next) y se repite el proceso.

Cuando no se encuentra el número:

* Si se recorre toda la lista y tmp llega a ser null (es decir, no hay más nodos), significa que el número no está en la lista. En este caso, se devuelve false.

**¿Qué son las listas doblemente enlazadas?**

Una **lista doblemente enlazada** es una estructura de datos en la que cada nodo está conectado tanto al nodo siguiente como al nodo anterior mediante punteros.

* **Estructura:** Cada nodo tiene tres partes:
  + **Datos:** El valor almacenado en el nodo.
  + **Puntero al nodo siguiente:** Apunta al próximo nodo en la lista.
  + **Puntero al nodo anterior:** Apunta al nodo anterior en la lista.
* **Ventajas:**
  + Puedes recorrer la lista en ambas direcciones (hacia adelante y hacia atrás).
  + Facilita operaciones como inserción y eliminación en cualquier posición.