Escuela de Minas
Dr. Horacio Carrillo

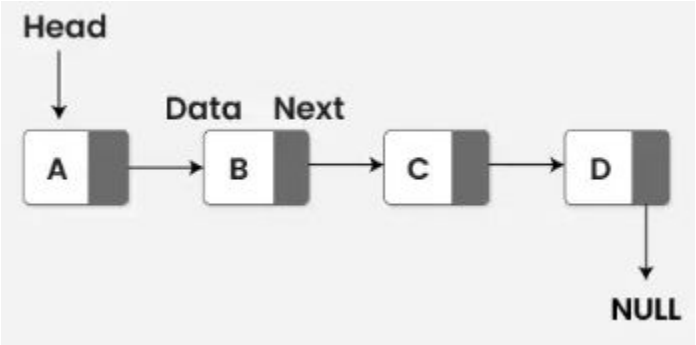
Estructura de datos

Unidad 5: TDA Listas simples

A.P.U. Gustavo Alberto Ordoñez

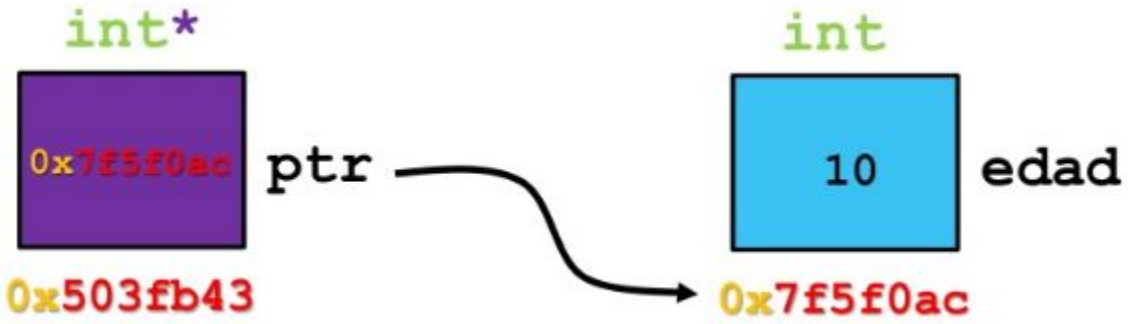
Concepto de lista enlazada

Una lista enlazada es una colección de elementos denominados nodos, dispuestos uno a continuación de otro, cada uno de ellos conectado al siguiente elemento por un enlace o referencia.



Tipo de dato puntero

Es un tipo de dato que indica la posición de memoria ocupada por otro dato, permitiendo que, durante la ejecución del programa, las estructuras dinámicas puedan cambiar sus tamaños. Tiene asociadas las operaciones de asignación y comparación de punteros.



Ejemplo de puntero en C++

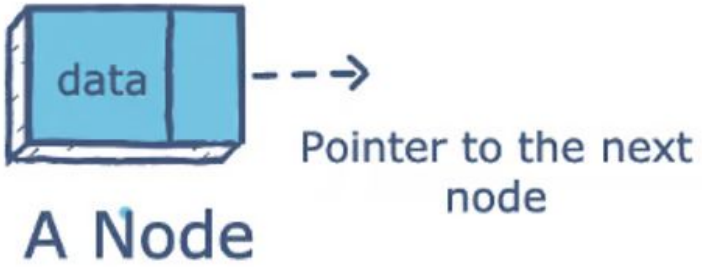
```
#include<iostream>
using namespace std;

int main(){
      int num = 17;
      int* punteroNum = nullptr;

      punteroNum = &num;

      cout << "Num: " << num << endl; // Muestra 17
      cout << "Puntero: " << punteroNum << endl; // Muestra una dirección de mem.
      cout << "Dirección de num: " << &num << endl; // Muestra una dirección de mem.
      return 0;
}
```

TDA Nodo

Un nodo es el componente elemental de una lista enlazada. Está compuesto por dos partes: El dato y el enlace.

Dato: Es el contendor del dato propiamente dicho.

Enlace: Es la dirección de memoria al siguiente nodo. El nodo que se encuentra al final de la lista no circular tiene valor nulo (NULL).



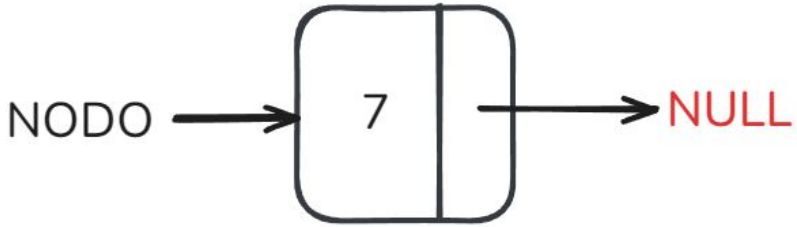
Operaciones con TDA Nodo

crear_nodo()

- **Objetivo:** Crear un nuevo nodo (se reserva memoria para un nuevo elemento)
- **Entrada:** un puntero a nodo.
- **Salida:** devuelve un puntero con la dirección del nodo creado. Si el nodo no puede crearse, retorna NULL o nulo.
- **Restricciones:** ninguna.

Implementación de un TDA Nodo en C++

```
typedef struct tnodo *pnodo;

//Implementación de nodo utilizando punteros
typedef struct tnodo{
    int dato;
    pnodo siguiente;
};
```

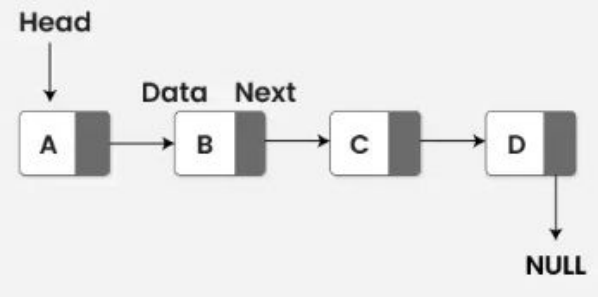


Clasificación

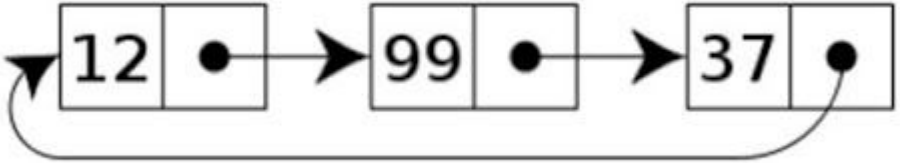
Las listas se pueden dividir en cuatro categorías:

- **Listas simple enlazadas:** Cada nodo contiene un único enlace que conecta ese nodo al nodo siguiente.
- **Listas dobles enlazadas:** Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor.
- **Listas circular simplemente enlazada:** Una lista simple en la que el último elemento se enlaza al primer elemento.
- **Lista circular doblemente enlazada:** Una lista doblemente enlazada en la que el último elemento se enlaza al primer elemento y viceversa.

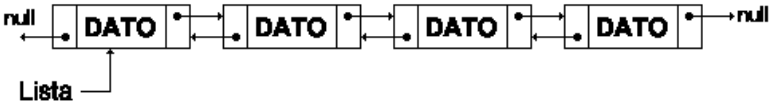
Ejemplos gráficos de los tipos de listas



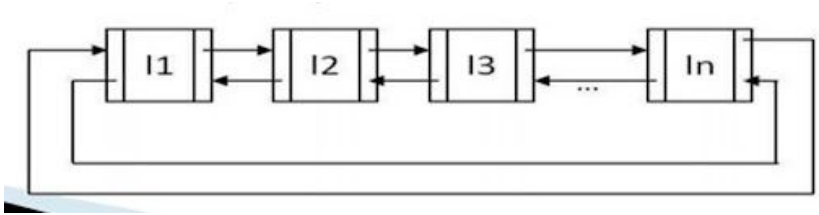
Lista simplemente enlazada



Lista circular simplemente enlazada



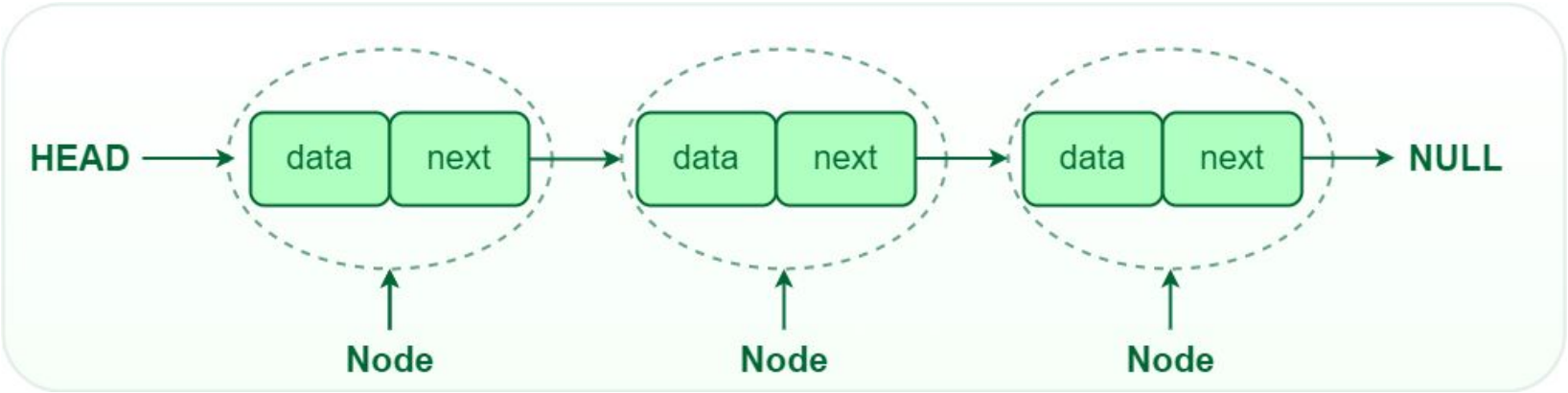
Lista doblemente enlazada



Lista circular doblemente enlazada

TDA Lista simplemente enlazada

Es una lista cuyos nodos contienen un **único enlace** que conecta al nodo siguiente o sucesor, excepto el nodo final que tiene un enlace **nulo** o **NULL**. Por lo que tiene una única dirección de recorrido hacia adelante.



Operaciones fundamentales

- iniciar_lista()
- es_vacia()
- agregar()
  - agregar_inicio()
  - agregar_final()
  - agregar_ordenado()
- buscar()
- eliminar()
  - eliminar_inicio
  - eliminar_nodo
  - eliminar_final
- mostrar_lista()

Iniciar_Lista()

Objetivo: inicializar la lista, generando una lista vacía.

Entrada: una lista (puntero de inicio de la lista).

Salida: devuelve una lista vacía (puntero de inicio de la lista en valor nulo).

Restricciones: ninguna

agregar_inicio()

Objetivo: agregar un nuevo nodo en el primer lugar de la lista.

Entrada: una lista y un nuevo dato.

Salida: devuelve una lista con un nuevo nodo al principio.

Restricciones: una lista inicializada y espacio en memoria para la creación del nuevo nodo.

agregar_final()

Objetivo: agregar un nuevo nodo al final de la lista.

Entrada: una lista y un nuevo dato.

Salida: devuelve una lista con un nuevo nodo al final.

Restricciones: una lista inicializada y espacio en memoria para la creación del nuevo nodo.

agregar_ordenado()

Objetivo: agregar un nuevo nodo respetando el orden de la lista.

Entrada: una lista y un nuevo dato.

Salida: devuelve una lista ordenada con un nuevo nodo.

Restricciones: una lista inicializada y espacio en memoria para la creación del nuevo nodo.

eliminar_inicio()

Objetivo: eliminar el primer nodo de la lista.

Entrada: una lista.

Salida: devuelve la lista con un nodo menos.

Restricciones: una lista inicializada, la lista no debe estar vacía.

eliminar_final()

Objetivo: eliminar el último nodo de la lista.

Entrada: una lista.

Salida: devuelve la lista con un nodo menos.

Restricciones: una lista inicializada, la lista no debe estar vacía.

eliminar_nodo()

Objetivo: eliminar un nodo específico de la lista.

Entrada: una lista y el valor a extraer.

Salida: devuelve la lista con un nodo menos o un mensaje de error en caso de no encontrar el elemento.

Restricciones: una lista inicializada, la lista no debe estar vacía.

mostrar_lista()

Objetivo: visualizar el contenido de los nodos de la lista.

Entrada: una lista (puntero de inicio de la lista).

Salida: se muestra por pantalla los datos almacenados en los nodos.

Restricciones: una lista inicializada, la lista no debe estar vacía.

buscar()

Objetivo: buscar un valor específico en la lista.

Entrada: una lista, el valor a buscar.

Salida: devuelve *true* si el dato está en la lista y *false* en caso contrario.

Restricciones: una lista inicializada, la lista no debe estar vacía.

Implementación de TDA lista

La creación de una lista enlazada requiere de una variable para almacenar la información y puntero de acceso a la lista, que apunta al primer nodo de la lista. Opcionalmente se puede declarar un puntero al nodo final de la lista.

```
typedef struct tnodo *pnodo;

typedef struct tnodo{
    int dato;
    pnodo siguiente;
};
```

- Donde *pnodo es un puntero al registro pnodo.
- int dato se utiliza para guardar datos de tipo entero.
- pnodo siguiente es el enlace al siguiente nodo.

Implementación crear_nodo()

```
void crear_nodo(pnodo &nuevo){
    nuevo = new tnodo;
    if(nuevo != NULL){
        cout << "Ingrese valor: ";
        cin >> nuevo -> dato;
        nuevo -> siguiente = NULL;
    }
    else{
        cout << "Memoria insuficiente" <<
endl;
    }
}
```

Se reserva espacio en memoria para crear un nodo.

Si hay espacio en memoria se crea el nodo con un nuevo dato.

En caso de no haber memoria suficiente se notifica al usuario.

Implementación de iniciar_lista()

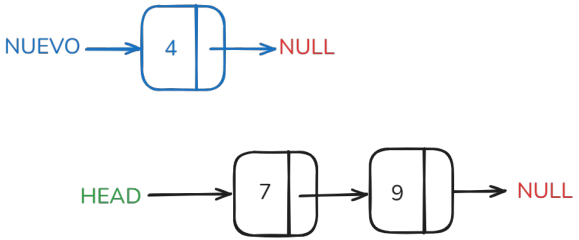
```
void iniciar_lista(pnodo &lista){
    lista = NULL;
}
```

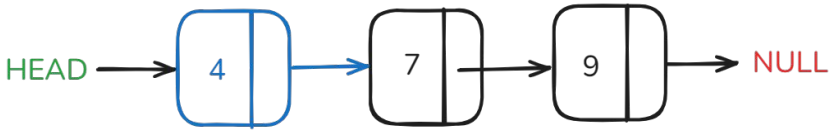
Se inicializa el puntero de acceso a la lista en NULL.

Implementación agregar_inicio()

```
void agregar_inicio(pnodo &lista, pnodo nuevo){
     nuevo -> siguiente = lista;
     lista = nuevo;
}
```

Se agrega al inicio



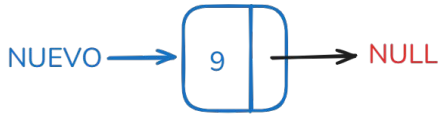


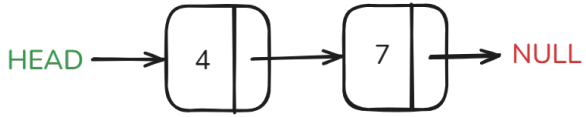
Implementación agregar_final

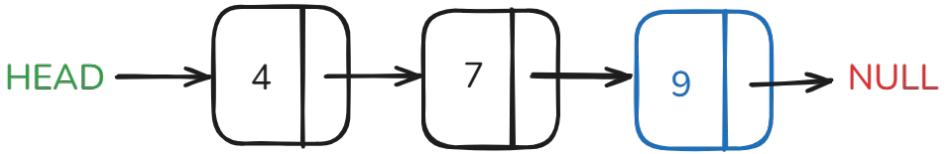
```
void agregar_final(pnodo &lista, pnodo nuevo){
    pnodo i;
    if(lista == NULL){
        lista = nuevo;
    }
    else{
        for(i = lista; i->siguiente != NULL;
i=i->siguiente);
        i->siguiente = nuevo;
    }
}
```

Si la lista está vacía se agrega al inicio.

Se recorre la lista hasta encontrar el último.







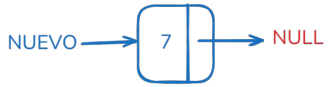
Implementación agregar_ordenado

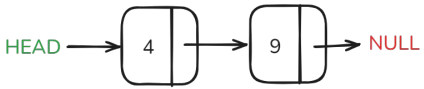
```
void agregar_ordenado(pnodo &lista, pnodo nuevo){
    pnodo i;
    if(lista == NULL){
        lista = nuevo;
    }
    else{
        if(nuevo->dato < lista->dato){
            nuevo->siguiente = lista;
            lista = nuevo;
        }
        else{
            for(i=lista; i->siguiente!=NULL && nuevo->dato > (i->siguiente)->dato; i=i->siguiente);
            nuevo->siguiente = i->siguiente;
            i->siguiente = nuevo;
        }
    }
}
```

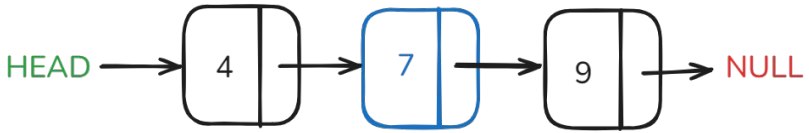
Si la lista está vacía se agrega al inicio.

Si el dato es menor al primero de la lista.

Se busca el lugar que le corresponde al dato.





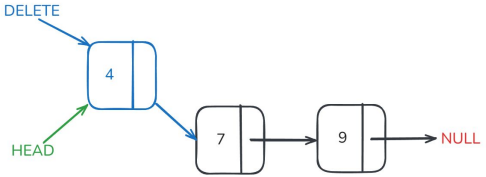


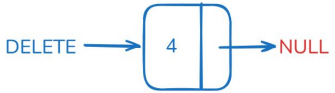
Implementación eliminar_inicio()

```
pnodo eliminar_inicio(pnodo &lista){
    pnodo borrado;
    if(lista==NULL){
        borrado = NULL;
    }
    else{
        borrado = lista;
        lista = lista->siguiente;
        borrado->siguiente = NULL;
    }
    return borrado;
}
```

Si la lista está vacía se retornará NULL

Si la lista no está vacía.





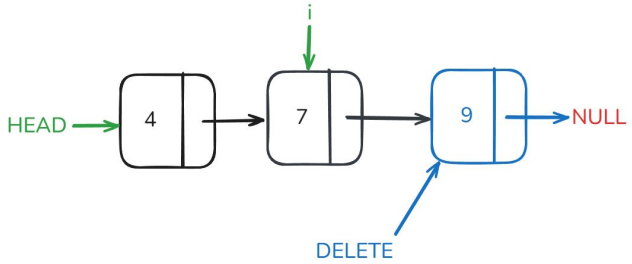
Implementación eliminar_final()

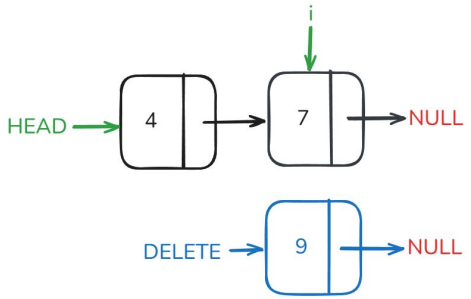
```
pnodo eliminar_final(pnodo &lista){
    pnodo borrado, i;
    if(lista == NULL){
        borrado = NULL;
    }
    else{
        if(lista->siguiente == NULL){
            borrado=lista;
            lista=NULL;
        }
        else{
            for(i=lista; (i->siguiente)->siguiente != NULL;
i=i->siguiente);
            borrado = i->siguiente;
            i->siguiente = NULL;
        }
    }
    return borrado;
}
```

Si la lista está vacía retornará NULL.

Si solo hay un elemento en la lista se elimina directamente.

Si hay más de un elemento en la lista.





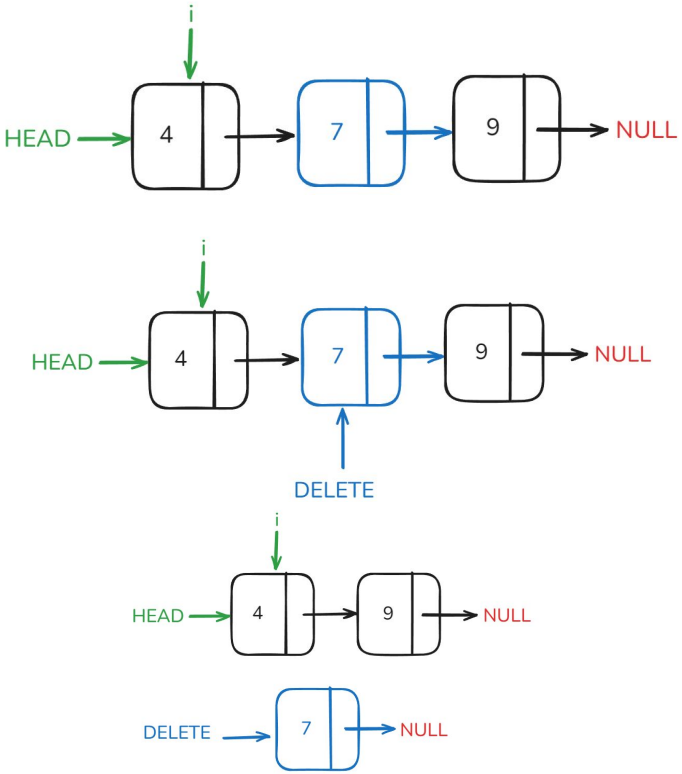
Implementación eliminar_nodo()

```
pnodo eliminar_nodo(pnodo &lista, int valor){
    pnodo borrado, i;
    if(lista == NULL){
        borrado = NULL;
    }
    else{
        if(lista->dato == valor){
            borrado = lista;
            lista = borrado->siguiente;
            borrado->siguiente = NULL;
        }
        else{
            for(i=lista; i->siguiente != NULL && valor!=(i->siguiente)->dato; i=i->siguiente);
            if(i->siguiente != NULL){
                borrado = i->siguiente;
                i->siguiente = borrado->siguiente;
                borrado->siguiente = NULL;
            }
            else{
                borrado=NULL;
            }
        }
    }
    return borrado;
}
```

Si la lista está vacía retorará NULL

Si el nodo a eliminar es el primero.

Se busca al nodo con el dato.



Implementación de mostrar()

```
void mostrar(pnodo lista){
    pnodo i;
    if(lista != NULL){
        for(i = lista; i != NULL; i = i->siguiente){
            cout <<  "Nodo: " << i->dato << endl;
        }
    }
    else{
        cout << "Lista vacía";
    }
}
```

Se muestra por pantalla el valor almacenado por cada nodo.

Implementación de buscar_nodo()

```
bool buscar_nodo(pnodo lista, int valor){
    pnodo i;
    bool encontrado = false;
    if(lista != NULL){
        for(i=lista; i != NULL && encontrado == false;
i=i->siguiente){
            if(i->dato == valor){
                encontrado = true;
            }
        }
    }
    return encontrado;
}
```

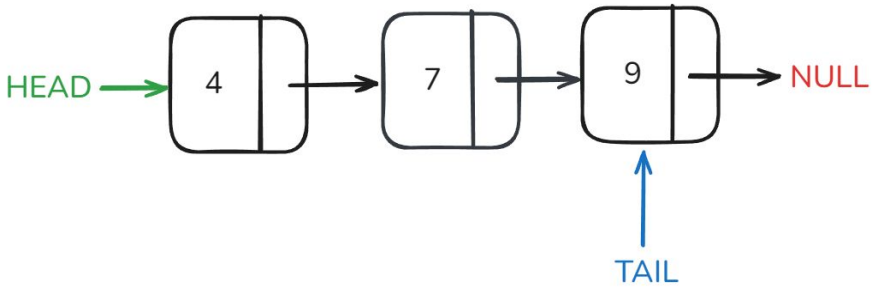
El bucle FOR deja de recorrer la lista una vez encontrado o cuando llega al final

Si la lista no está vacía se busca el nodo con el dato indicado.

Implementación de lista con dos enlaces

TDA Lista implementado del tipo nodo y el puntero a este inicio es un puntero que indica el primer nodo de la lista final es un puntero que indica el último nodo de la lista

```
typedef struct tnodo *pnodo;

typedef struct tnodo{
    int dato;
    pnodo siguiente;
};

typedef struct tlista{
    pnodo inicio;
    pnodo final;
};
```



Implementación de iniciar_lista()

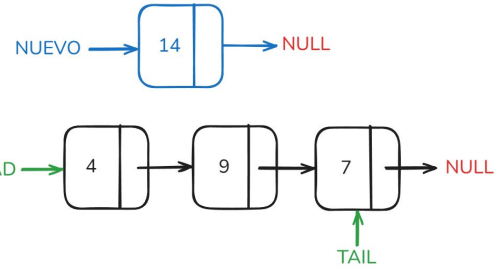
```
void iniciar_lista(tlista &lista){
    lista.inicio = NULL;
    lista.final = NULL;
}
```

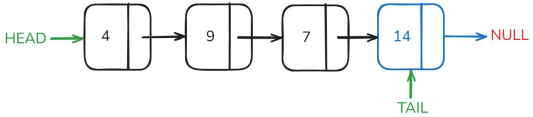
Implementación de agregar_final

```
void agregar_final(tlista &lista, pnodo nuevo){
    if(lista.inicio == NULL){
        lista.inicio = nuevo;
        lista.final = nuevo;
    }
    else{
        lista.final->siguiente = nuevo;
        lista.final = nuevo;
    }
}
```

Si la lista está vacía.

La lista no está vacía.





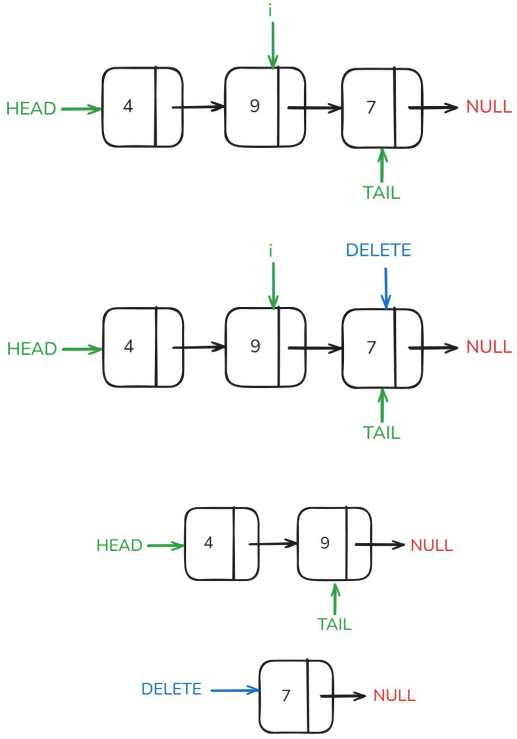
Implementación de eliminar_final()

```
pnodo eliminar_final(tlista &lista){
    pnodo borrado, i;
    if(lista.inicio == NULL){
        borrado = NULL;
    }
    else{
        if(lista.inicio == lista.final){
            borrado = lista.inicio;
            lista.inicio = NULL;
            lista.final = NULL;
        }
        else{
            for(i = lista.inicio; (i->siguiente)->siguiente != NULL;
i=i->siguiente);
            borrado = lista.final;
            lista.final = i;
            lista.final->siguiente = NULL;

        }
    }
    return borrado;
}
```

Si la lista está vacía

Si hay un único nodo en la lista

Hay más de un nodo en la lista



En resumen

- Los componentes de una lista están ordenados por sus campos de enlace en vez de ordenados físicamente como están en un array.
- El final de la lista se señala mediante una constante o puntero especial llamado NULL.
- Una lista simplemente enlazada contiene sólo un enlace a un sucesor único, a menos que sea el último, en cuyo caso no se enlaza con ningún otro nodo.
- Cuando se inserta un elemento en una lista enlazada, se deben considerar cuatro casos: ***añadir a una lista vacía***, **a*ñadir al principio de la lista*, *añadir en el interior*** y ***añadir al final de la lista***.
- Para borrar un elemento, primero hay que buscar el nodo que lo contiene y considerar dos casos: ***borrar el primer nodo***, **borrar un nodo en el medio y borrar nodo final** de la lista.
- El recorrido de una lista enlazada significa pasar por cada nodo (visitar) y procesarlo.

Bibliografía

Joyanes Aguilar. Estructuras de datos en C++. McGraw Hill. 2007.

Hernandez Roberto. Estructuras de datos y algoritmos. Prentice Hall. 2001.

Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.