



Escuela de
Ciencia y Tecnología
ECyT_UNSAM

Programación (en C)

Primer Cuatrimestre 2025

programacionbunsam@gmail.com



Listas Enlazadas



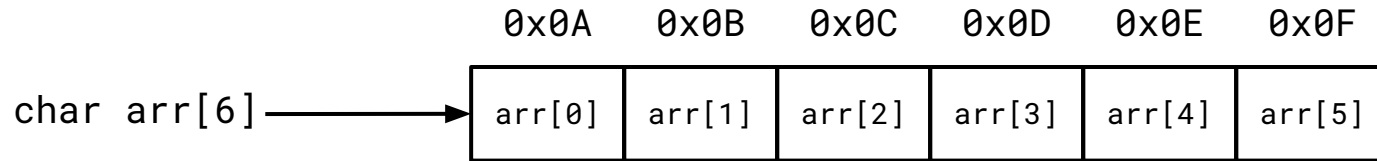
Listas Enlazadas - definición

Las listas enlazadas son estructuras de datos lineales cuyos elementos no están almacenados en direcciones de memoria contiguas, sino que están enlazados mediante punteros.



Listas Enlazadas - definición

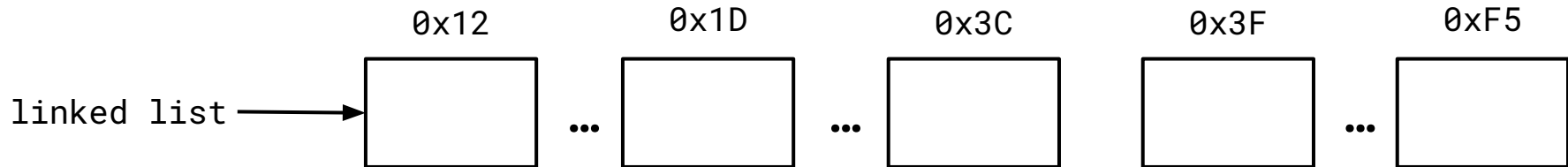
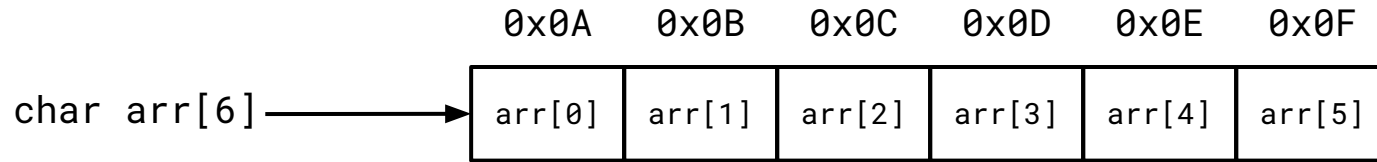
Las listas enlazadas son estructuras de datos lineales cuyos elementos no están almacenados en direcciones de memoria contiguas, sino que están enlazados mediante punteros.





Listas Enlazadas - definición

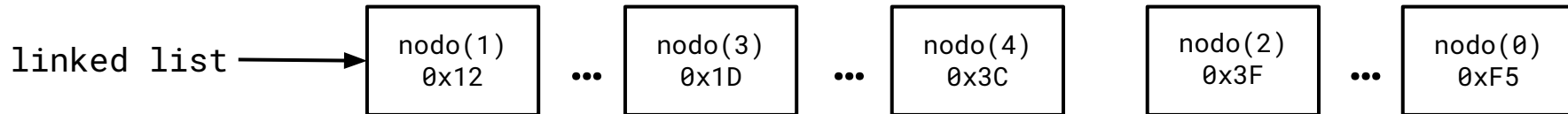
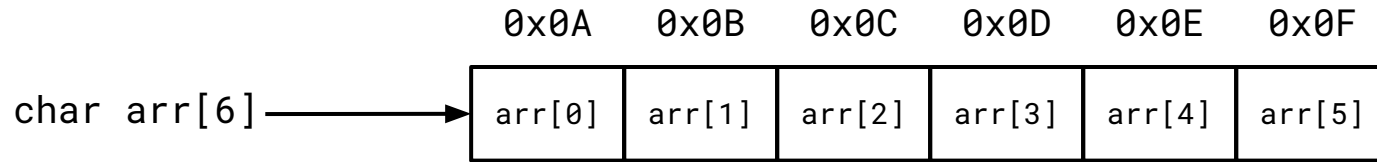
. Las listas enlazadas son estructuras de datos lineales cuyos elementos no están almacenados en direcciones de memoria contiguas, sino que están enlazados mediante punteros.





Listas Enlazadas - definición

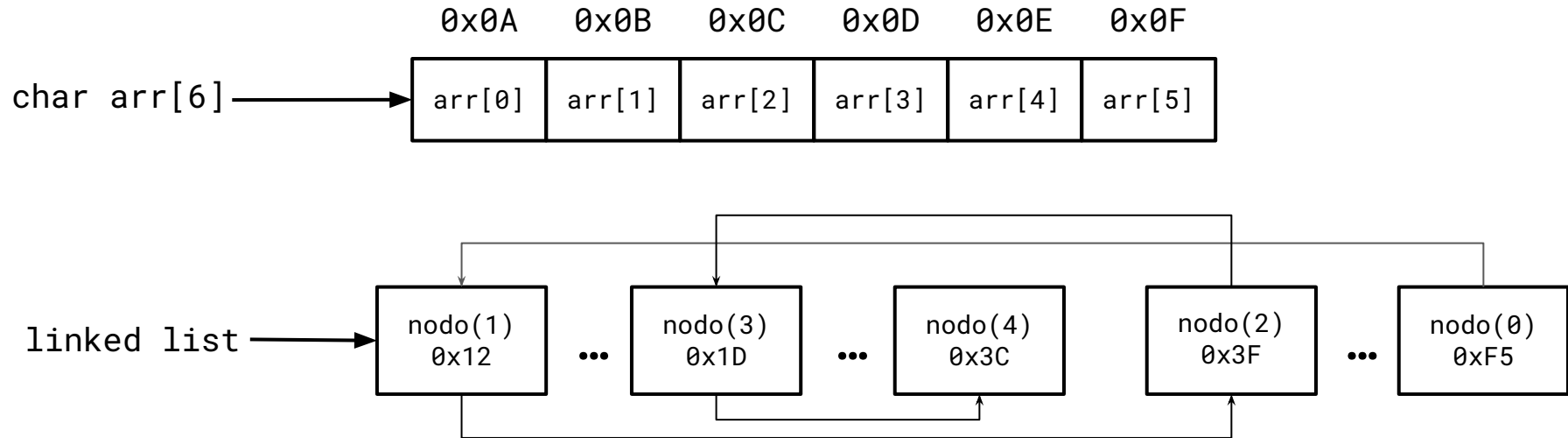
. Las listas enlazadas son estructuras de datos lineales cuyos elementos no están almacenados en direcciones de memoria contiguas, sino que están enlazados mediante punteros.





Listas Enlazadas - definición

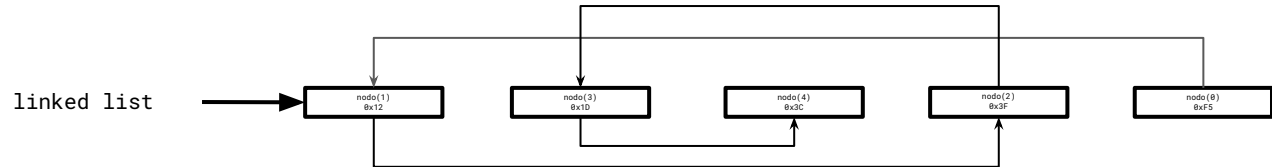
. Las listas enlazadas son estructuras de datos lineales cuyos elementos no están almacenados en direcciones de memoria contiguas, sino que están enlazados mediante punteros.





Listas Enlazadas - Nodo

def. Son el bloque constitutivo de las listas enlazadas. Podemos distinguir dos 'tipos' de miembros dentro de cada nodo:

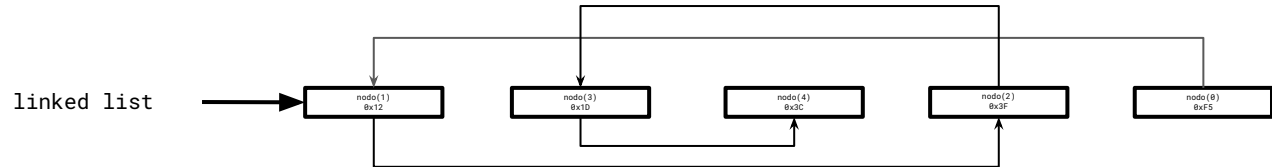




Listas Enlazadas - Nodo

def. Son el bloque constitutivo de las listas enlazadas. Podemos distinguir dos 'tipos' de miembros dentro de cada nodo:

- Aquellos que contienen información
- Aquellos que apuntan a otro nodo de la lista



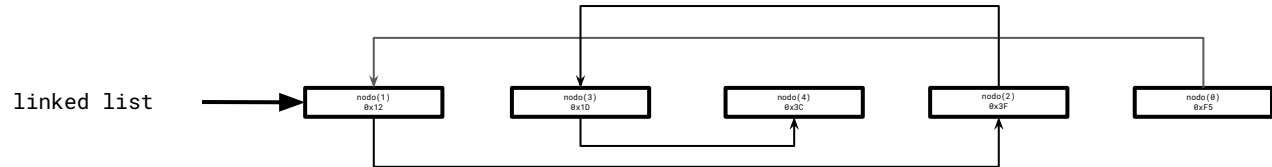


Listas Enlazadas - Nodo

```
typedef struct{
    int data;
    struct node_t *next;
}node_t;
```

def. Son el bloque constitutivo de las listas enlazadas. Podemos distinguir dos 'tipos' de miembros dentro de cada nodo:

- Aquellos que contienen información
- Aquellos que apuntan a otro nodo de la lista



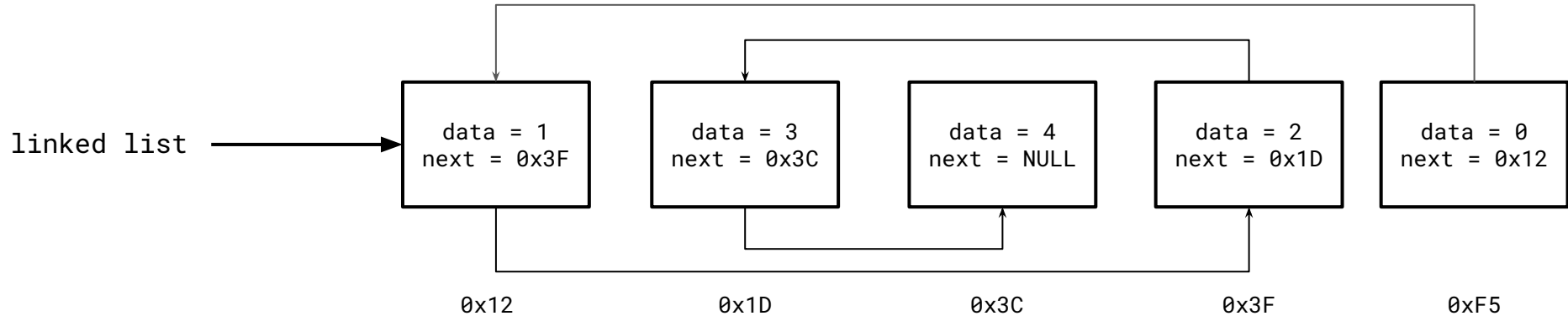


Listas Enlazadas - Nodo

```
typedef struct{  
    int data;  
    struct node_t *next;  
}node_t;
```

def. Son el bloque constitutivo de las listas enlazadas. Podemos distinguir dos 'tipos' de miembros dentro de cada nodo:

- Aquellos que contienen información
- Aquellos que apuntan a otro nodo de la lista





Principio y Fin

Como no conozco (a priori) la cantidad de elementos, necesito alguna manera de saber donde comienza y donde termina la lista.

```
typedef struct{  
    int data;  
    struct node_t *next;  
}node_t;
```



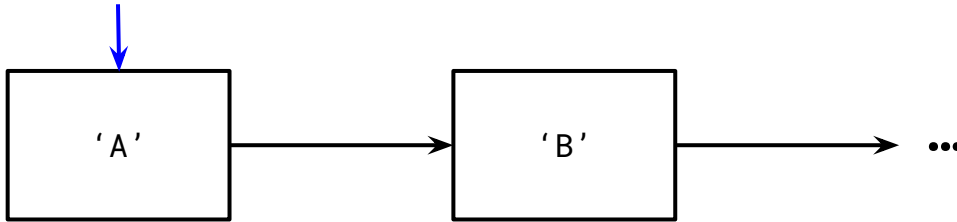
Principio y Fin

Como no conozco (a priori) la cantidad de elementos, necesito alguna manera de saber donde comienza y donde termina la lista.

→ El principio será llamado “head” o cabeza de la lista. Para conocerla, deberemos guardar la dirección de memoria en alguna variable.

```
typedef struct{  
    int data;  
    struct node_t *next;  
}node_t;
```

```
node_t* head = &firstNode;
```





Principio y Fin

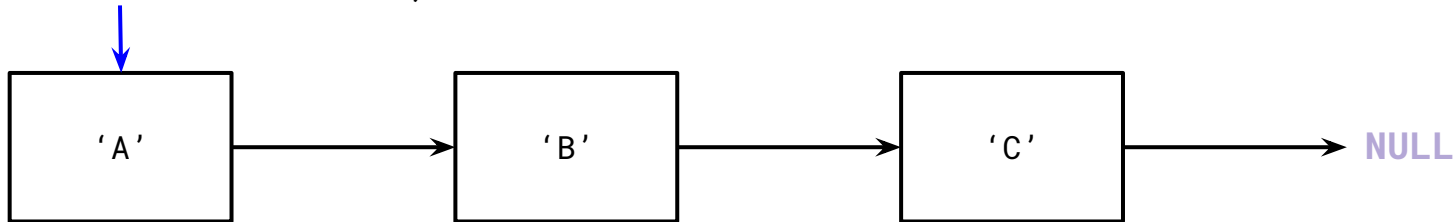
```
typedef struct{  
    int data;  
    struct node_t *next;  
}node_t;
```

Como no conozco (a priori) la cantidad de elementos, necesito alguna manera de saber donde comienza y donde termina la lista.

→ El principio será llamado “head” o cabeza de la lista. Para conocerla, deberemos guardar la dirección de memoria en alguna variable.

→ El final de la lista será cuando ya no haya un próximo nodo. Cuando esto ocurra, ‘next’ apuntará a NULL.

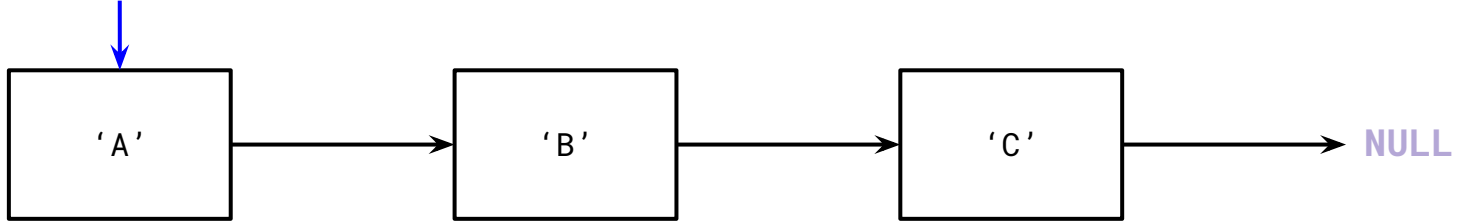
```
node_t* head = &firstNode;
```





Insertar elementos - principio

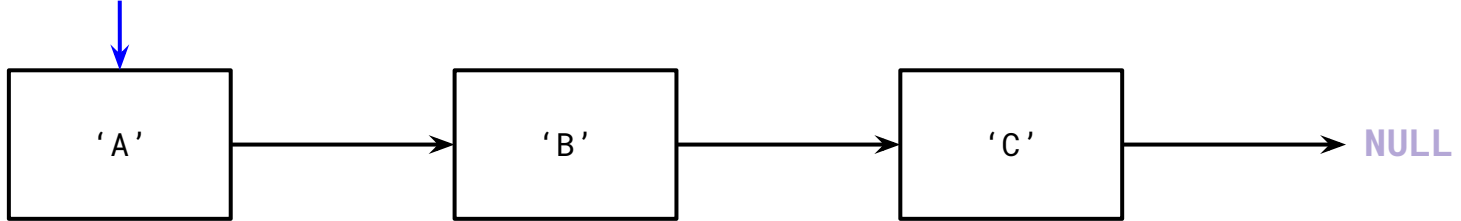
```
node_t* head = &firstNode;
```



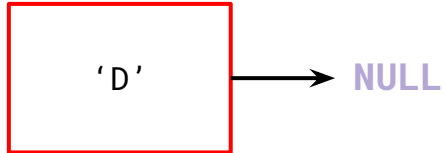


Insertar elementos - al principio

```
node_t* head = &firstNode;
```



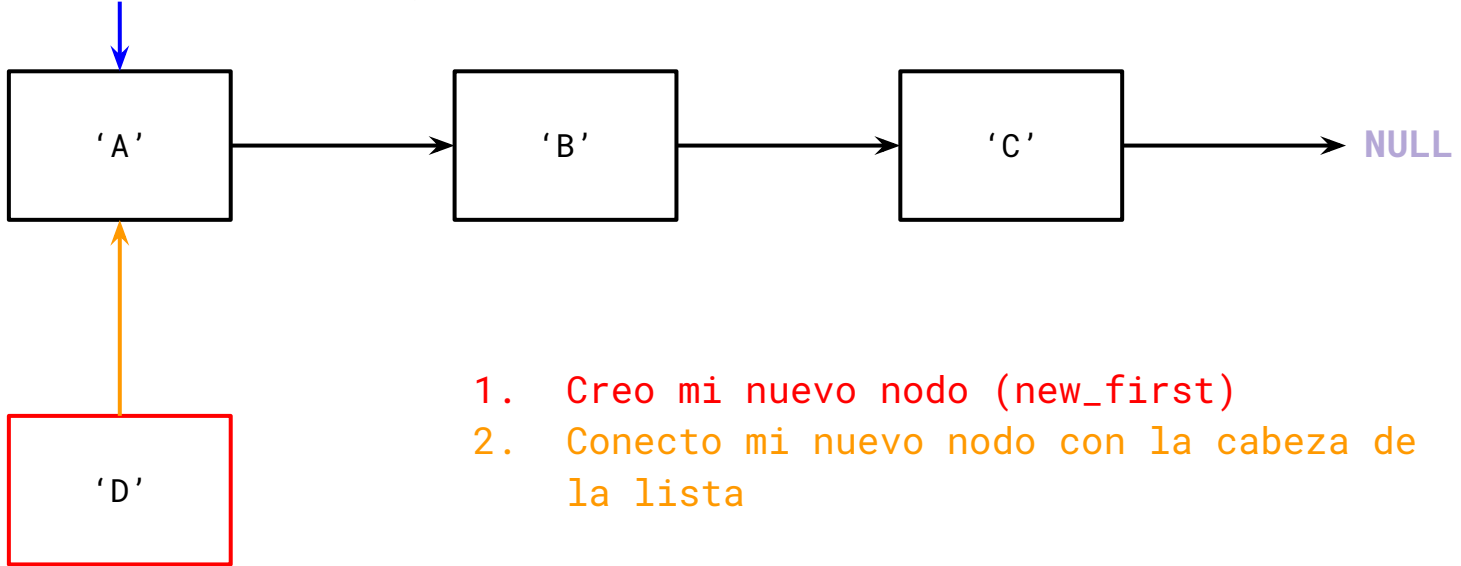
1. Creo mi nuevo nodo (new_first)





Insertar elementos - al principio

```
node_t* head = &firstNode;
```

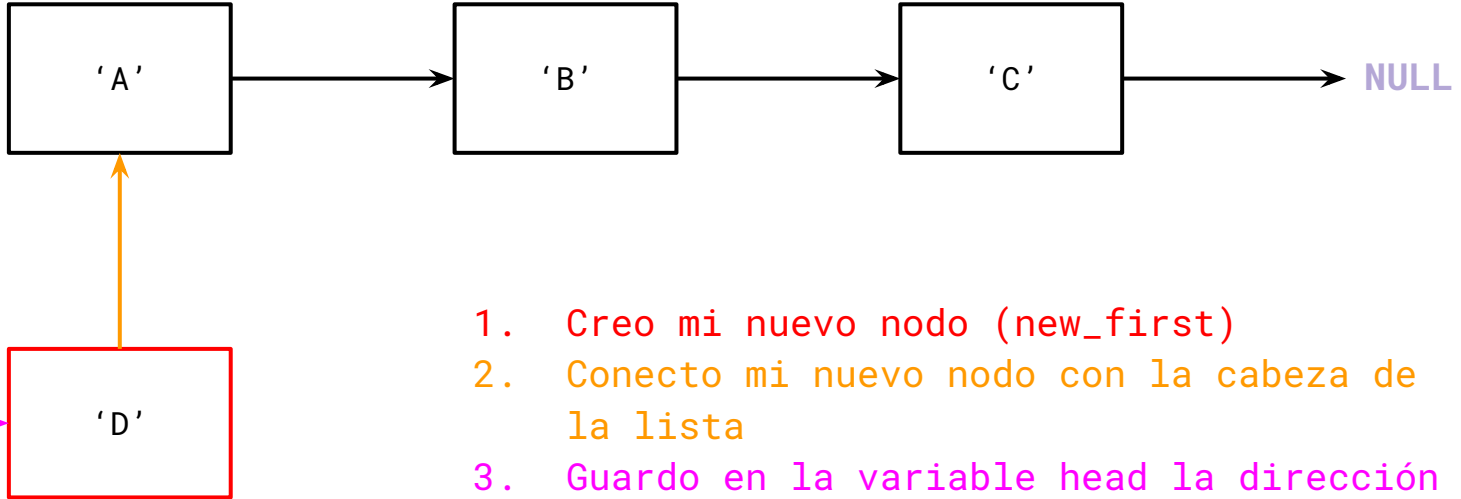


1. Creo mi nuevo nodo (new_first)
2. Conecto mi nuevo nodo con la cabeza de la lista



Insertar elementos - al principio

```
node_t* head = &new_first;
```

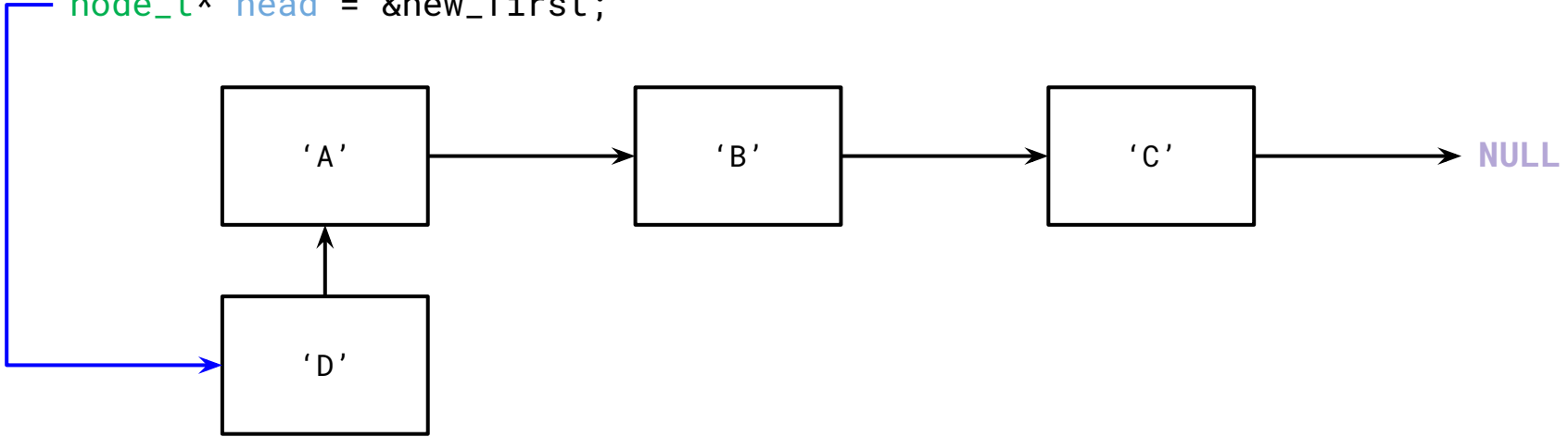


1. Creo mi nuevo nodo (new_first)
2. Conecto mi nuevo nodo con la cabeza de la lista
3. Guardo en la variable head la dirección del nuevo nodo



Insertar elementos - al final

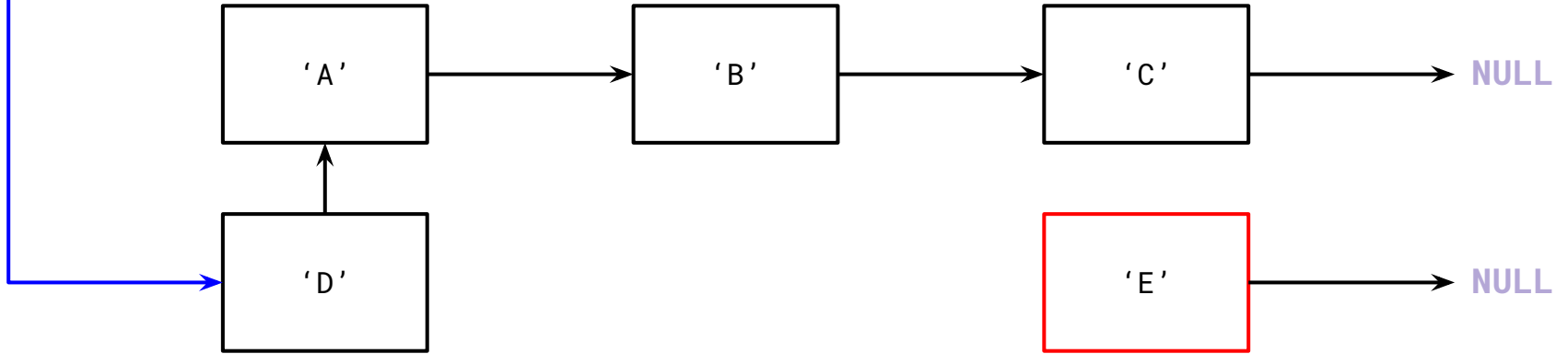
```
node_t* head = &new_first;
```





Insertar elementos - al final

```
node_t* head = &new_first;
```

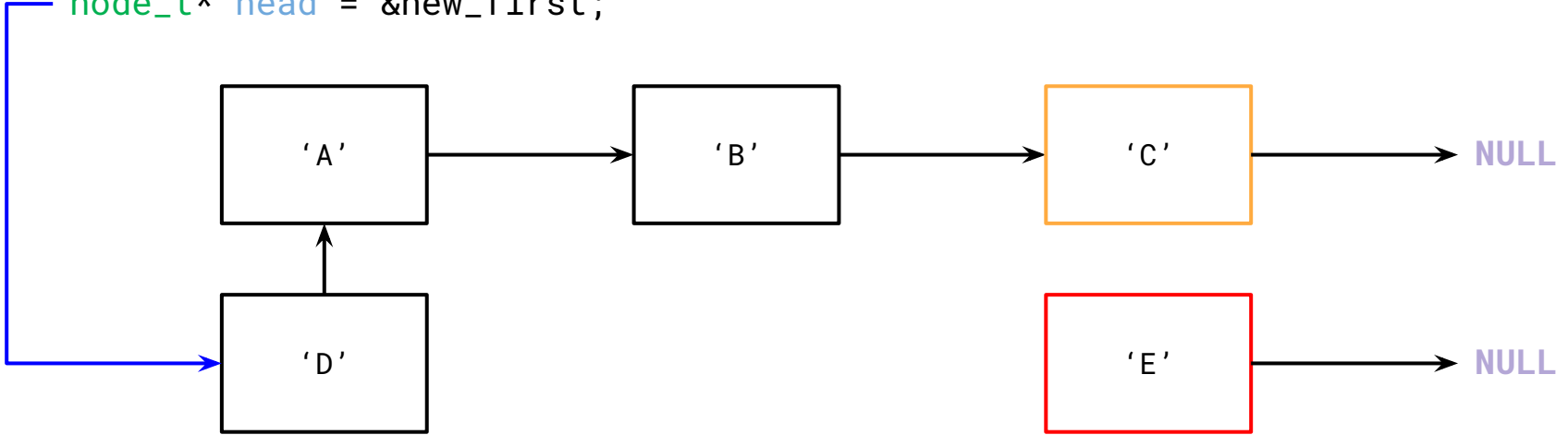


1. Creo mi nuevo nodo (new_last)



Insertar elementos - al final

```
node_t* head = &new_first;
```

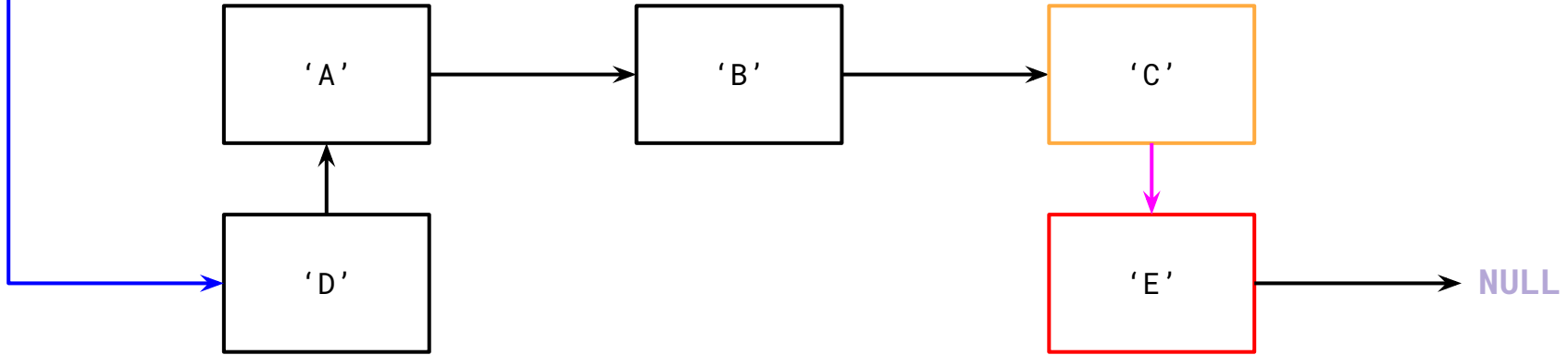


1. Creo mi nuevo nodo (new_last)
2. Recorro la lista hasta el final



Insertar elementos - al final

```
node_t* head = &new_first;
```

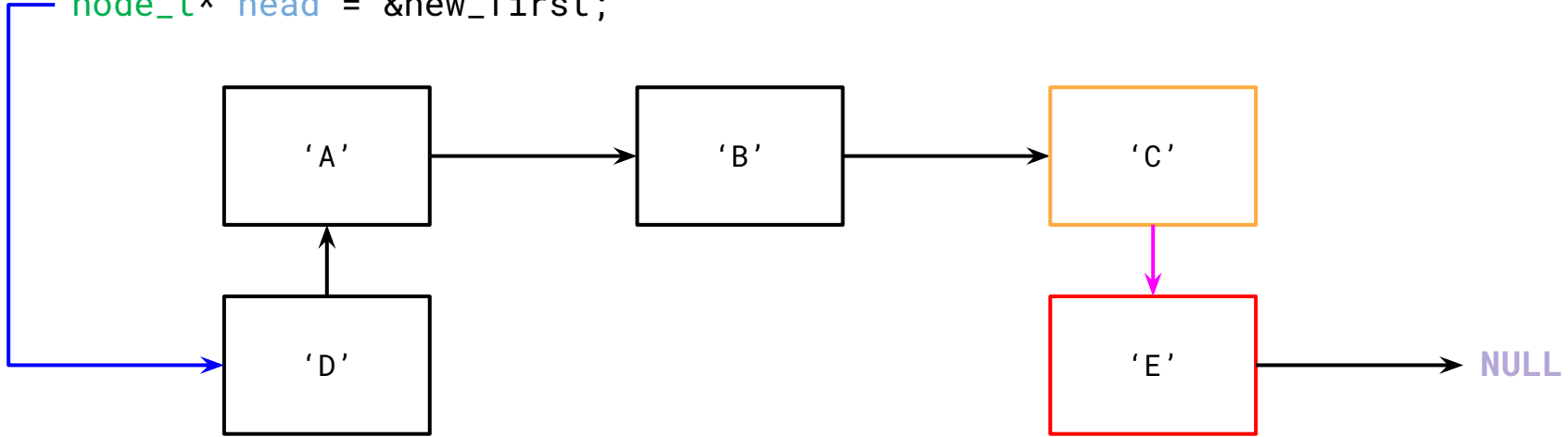


1. Creo mi nuevo nodo (new_last)
2. Recorro la lista hasta el último nodo
3. Conecto el último nodo con el nuevo nodo (new_last)



Insertar elementos - al final

```
node_t* head = &new_first;
```

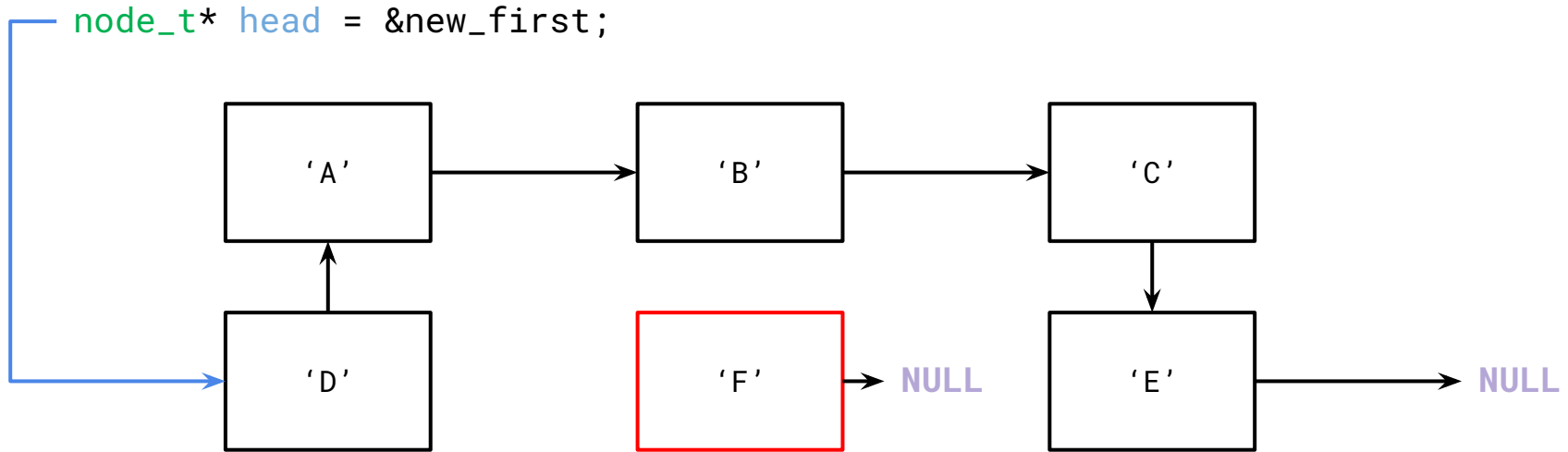


1. Creo mi nuevo nodo (new_last)
2. Recorro la lista hasta el último nodo
3. Conecto el último nodo con el nuevo nodo (new_last)

obs. Al crear un nodo, uno siempre inicializa el miembro 'next' en NULL, por lo que no necesito modificarlo al insertar new_last en la lista



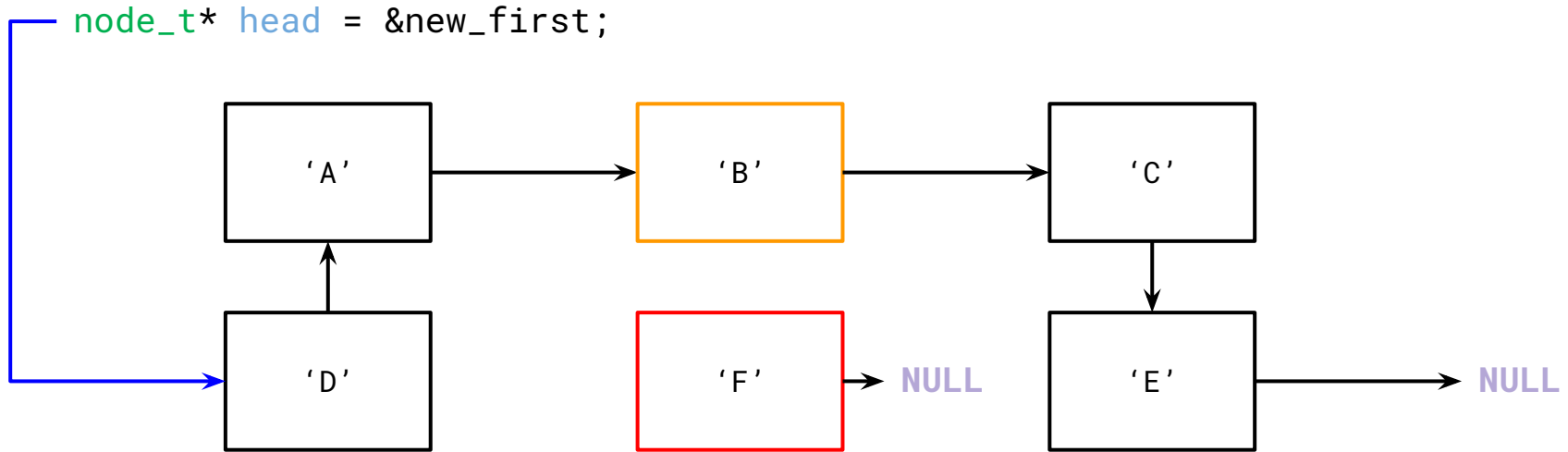
Insertar elementos - en el medio



1. Creo mi nuevo nodo (new_node)



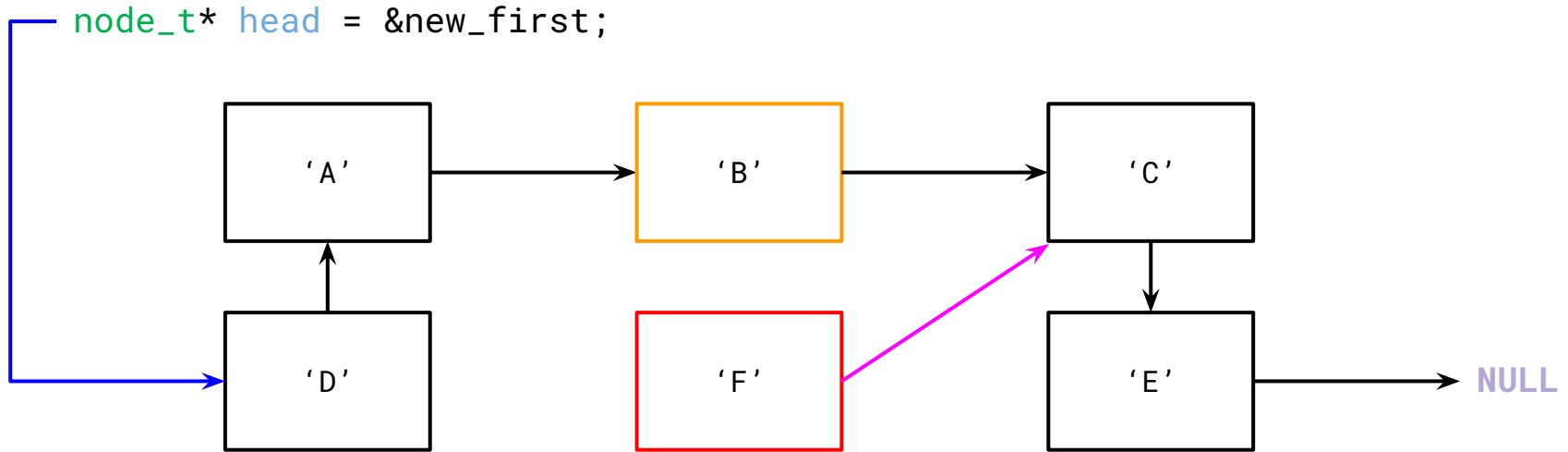
Insertar elementos - en el medio



1. Creo mi nuevo nodo (new_node)
2. Recorro hasta donde quiero insertar



Insertar elementos - en el medio

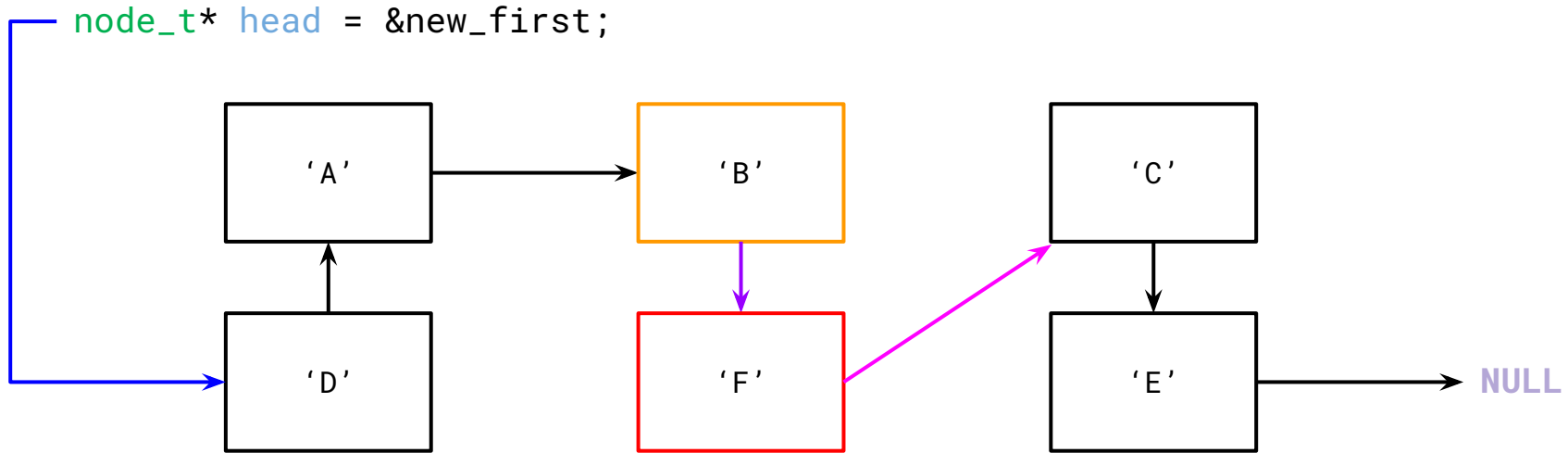


1. Creo mi nuevo nodo (new_node)
2. Recorro hasta donde quiero insertar

3. Conecto mi nuevo nodo con el que va a ser su próximo



Insertar elementos - en el medio



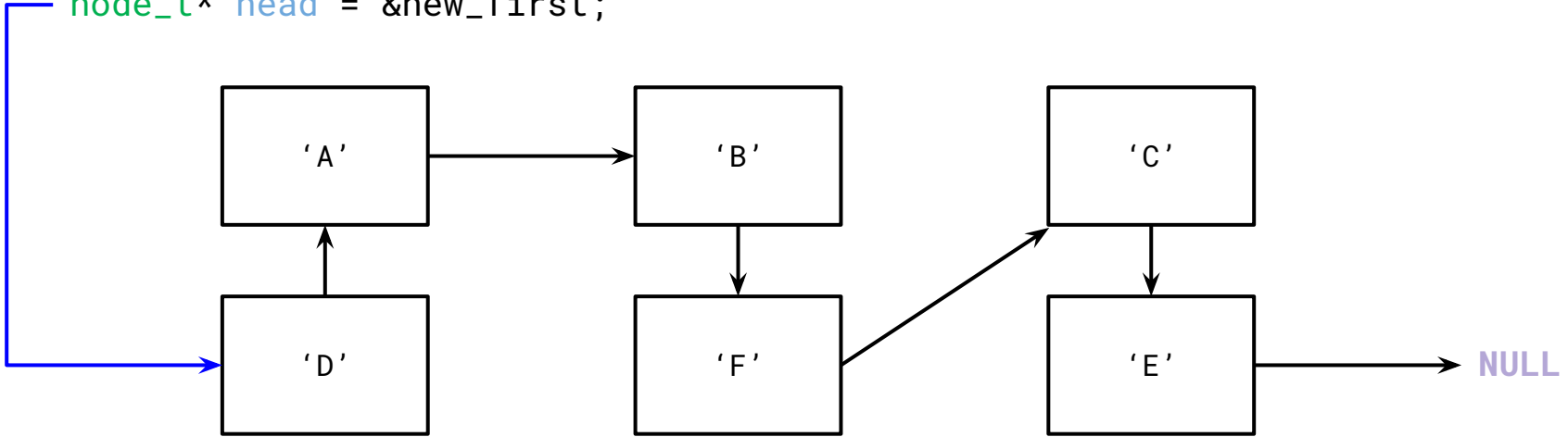
1. Creo mi nuevo nodo (new_node)
2. Recorro hasta donde quiero insertar

3. Conecto mi nuevo nodo con el que va a ser su próximo
4. Modifico 'next' del nodo naranja



Eliminar elementos - al principio

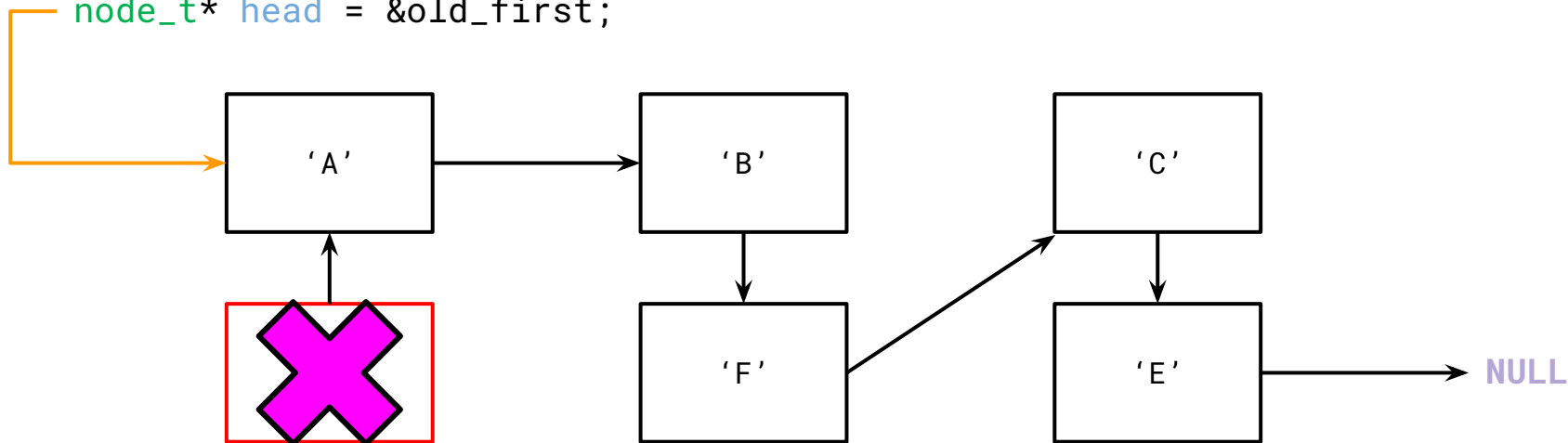
```
node_t* head = &new_first;
```





Eliminar elementos - al principio

```
node_t* head = &old_first;
```

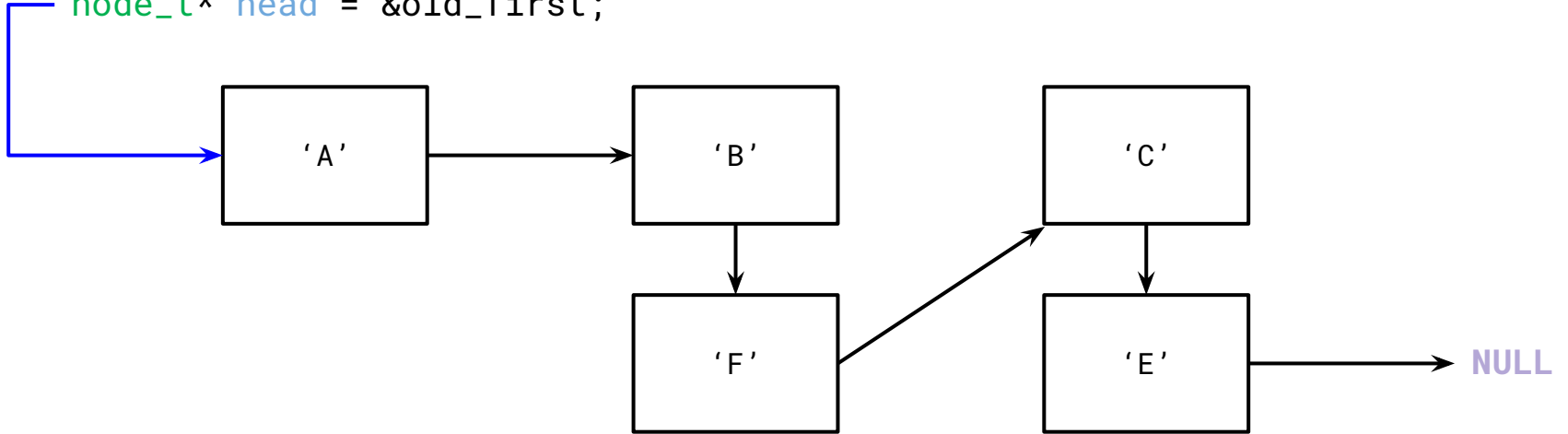


1. Guardo la dirección de memoria del primer nodo en una variable auxiliar (temp)
2. Muevo el lugar al que apunta head
3. Libero el que era el primer nodo (temp)



Eliminar elementos - al final

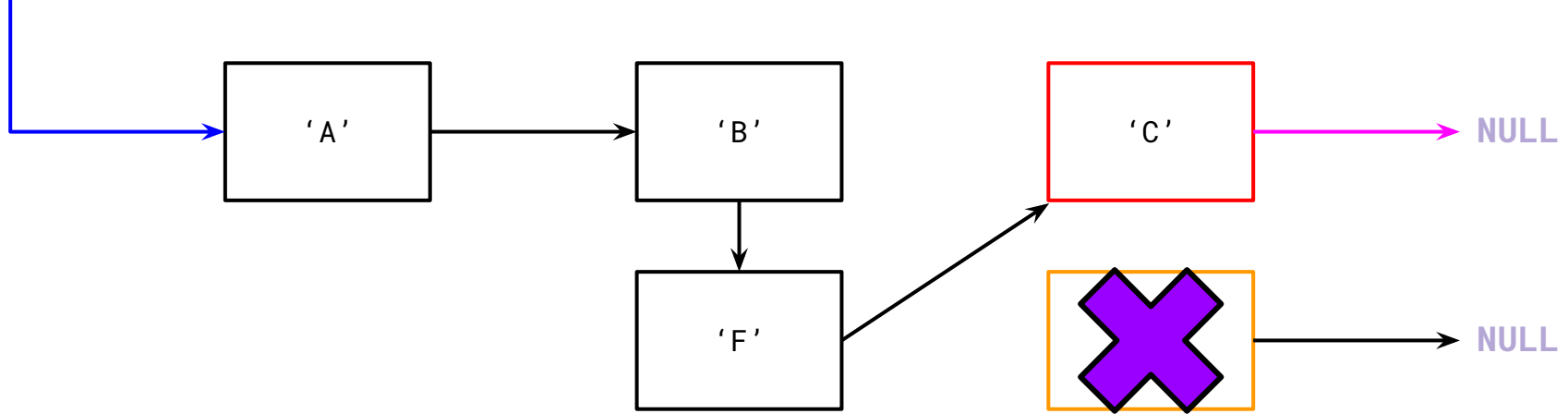
```
node_t* head = &old_first;
```





Eliminar elementos - al final

```
node_t* head = &old_first;
```



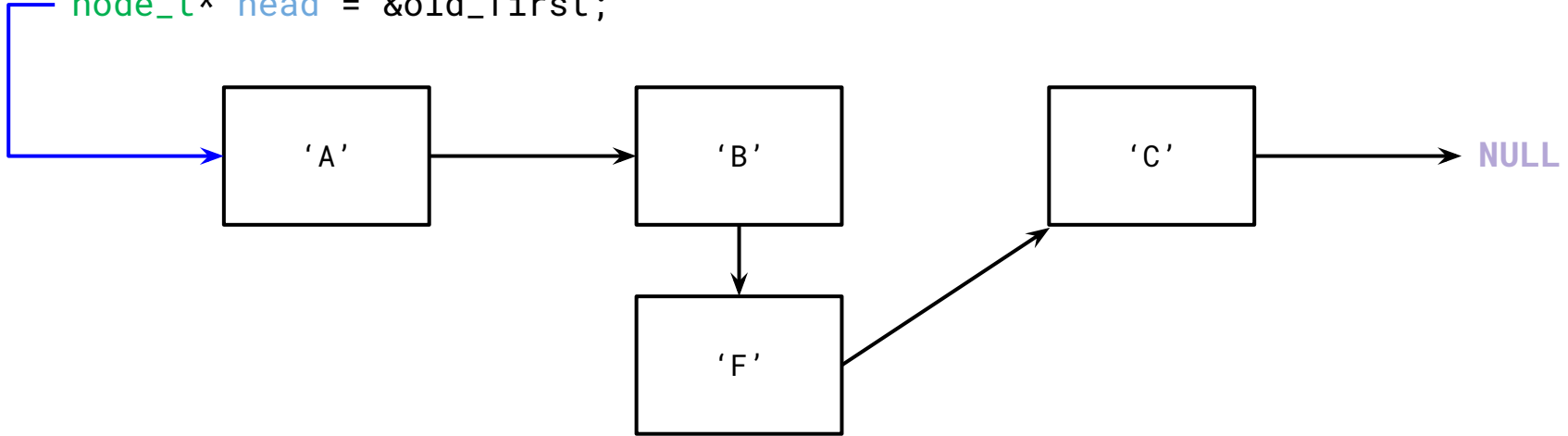
1. Recorro hasta el ante último
2. Guardo el último nodo en una variable temporal (temp)

3. Modifico para que el ante-último apunte a NULL
4. Libero el último nodo (temp)



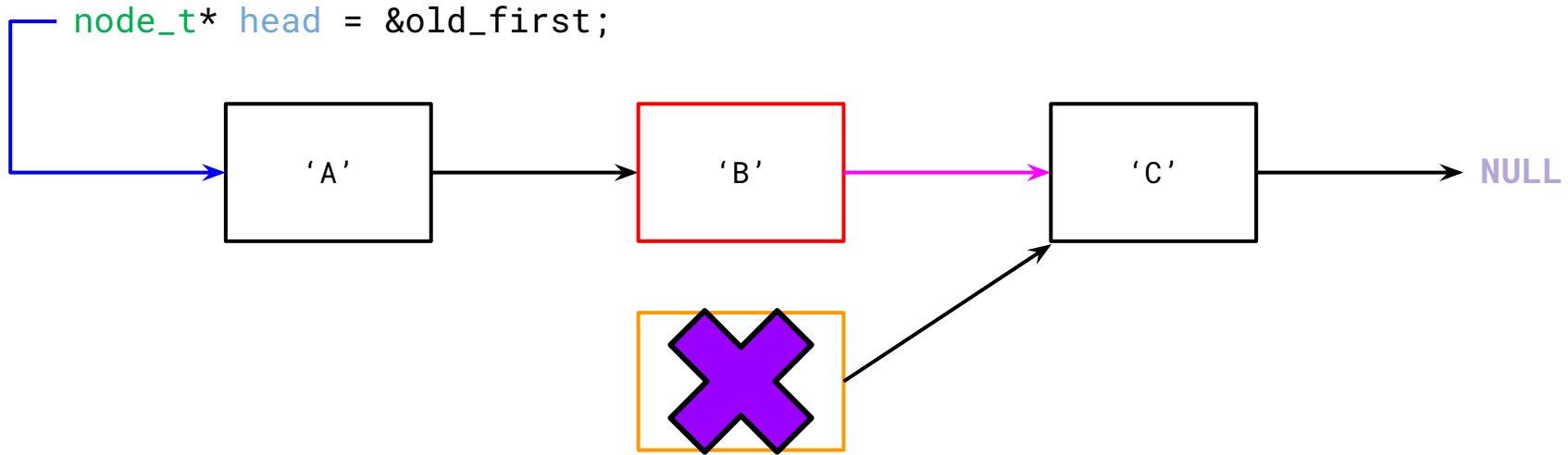
Eliminar elementos - en el medio

```
node_t* head = &old_first;
```





Eliminar elementos - en el medio

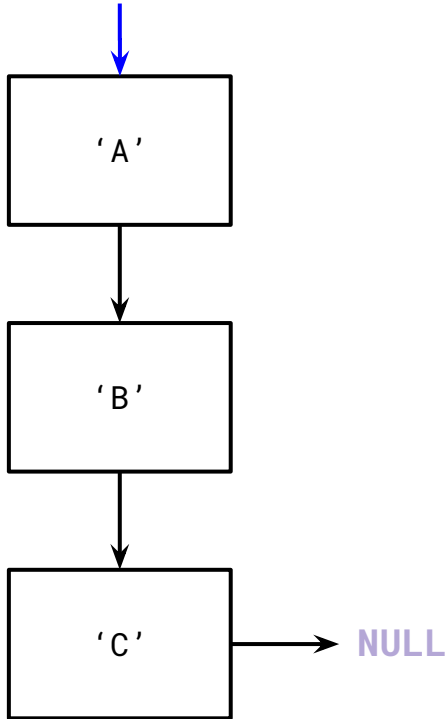


1. Recorro hasta el nodo anterior al que quiero eliminar
2. Guardo el nodo a eliminar en una variable auxiliar (temp)
3. Modifico las conexiones para saltarme el nodo que quiero eliminar
4. Libero el nodo que guarde en temp



Implementaciones - Pilas (Stack)

```
node_t* head = &first_node;
```

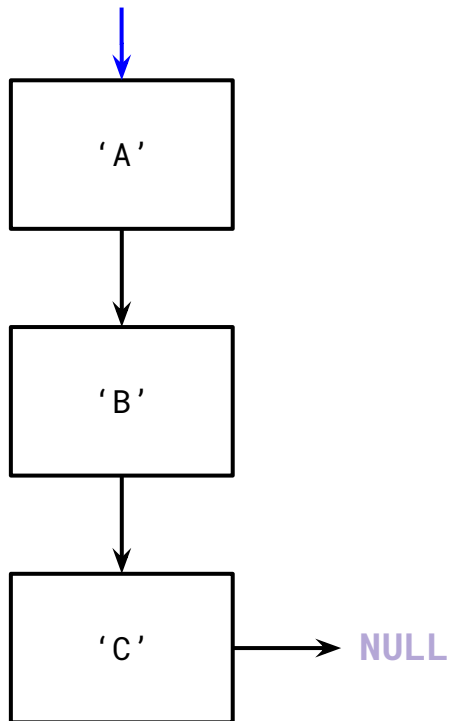


Dijimos (Clase 5) que las pilas son estructuras lineales que siguen el principio LIFO (Last In First Out) y que todas las operaciones ocurren en la cabeza.



Implementaciones - Pilas (Stack)

```
node_t* head = &first_node;
```

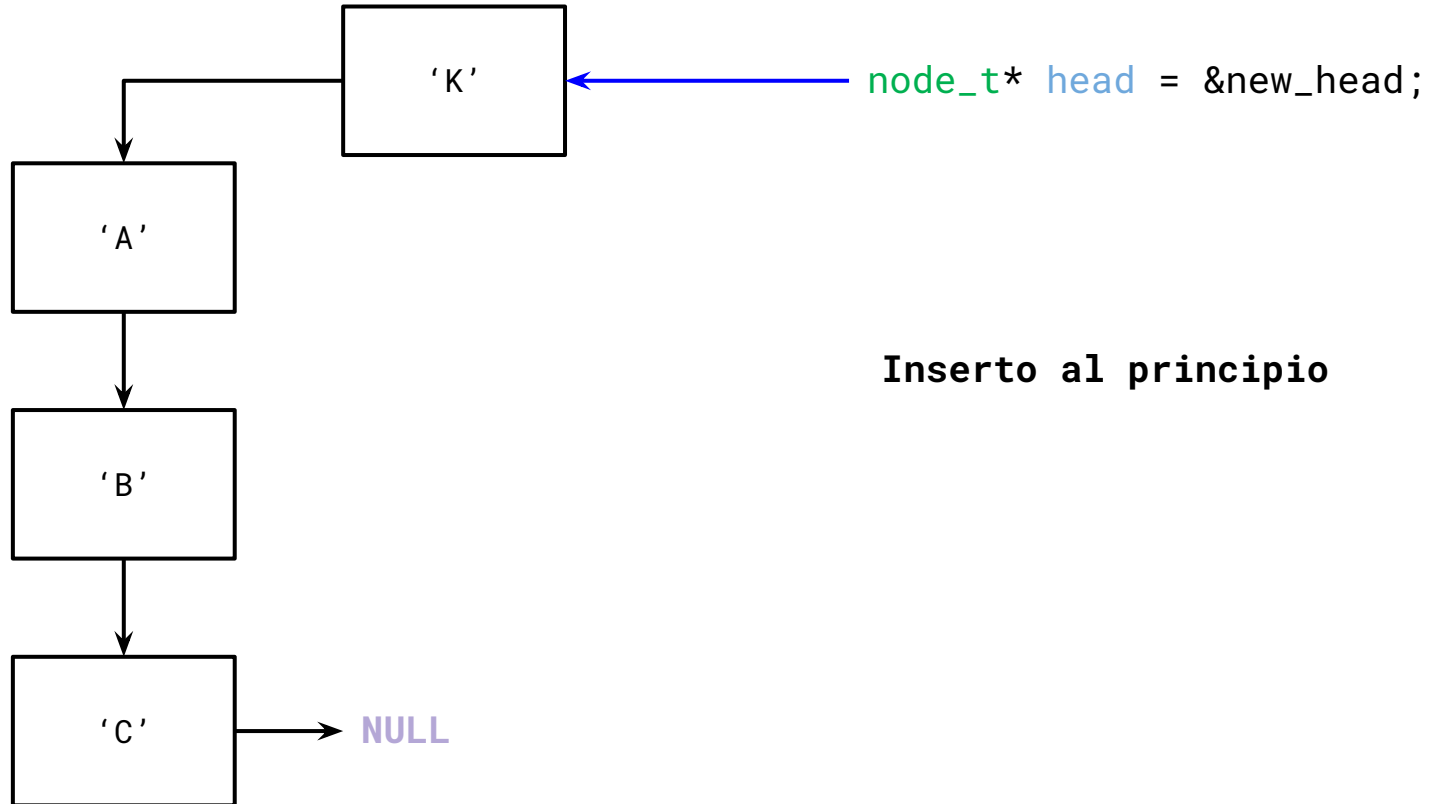


Dijimos (Clase 5) que las pilas son estructuras lineales que siguen el principio LIFO (Last In First Out) y que todas las operaciones ocurren en la cabeza.

Para implementarlas, entonces, simplemente tengo que usar lo que vimos recién: **inserto elementos al principio y elimino elementos al principio**

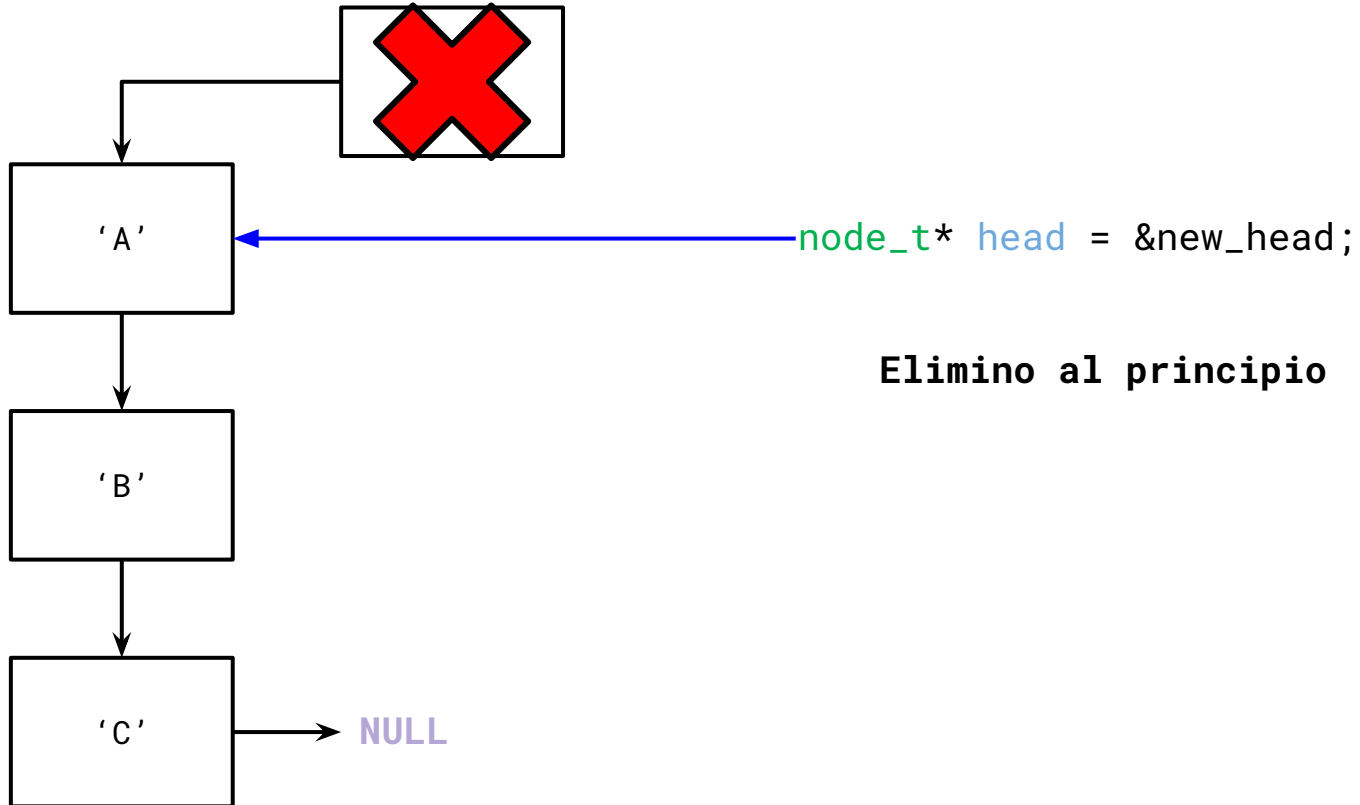


Implementaciones - Pilas (Stack)



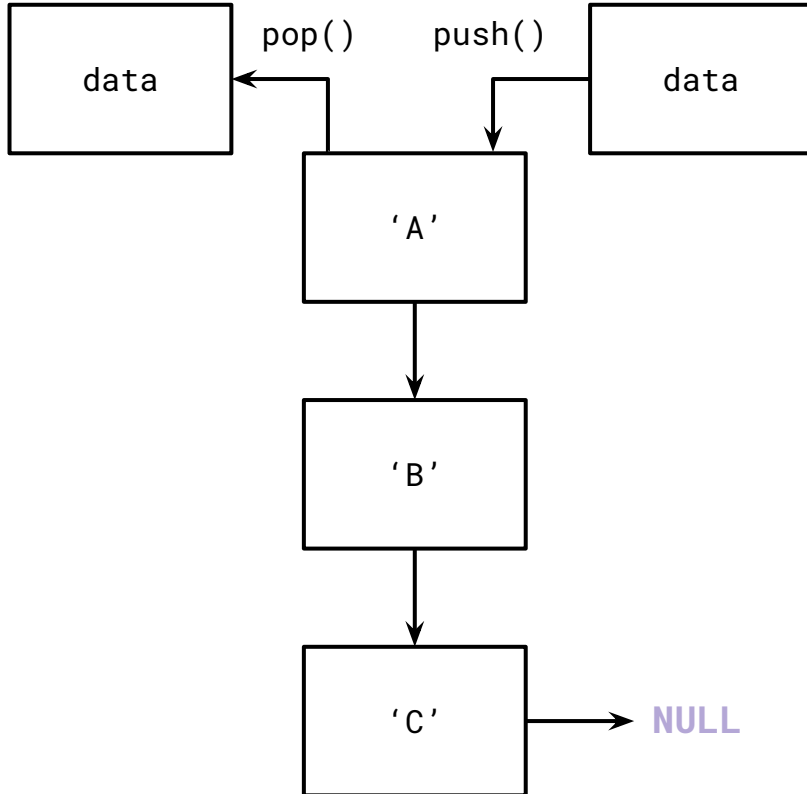


Implementaciones - Pilas (Stack)





Implementaciones - Pilas (Stack)



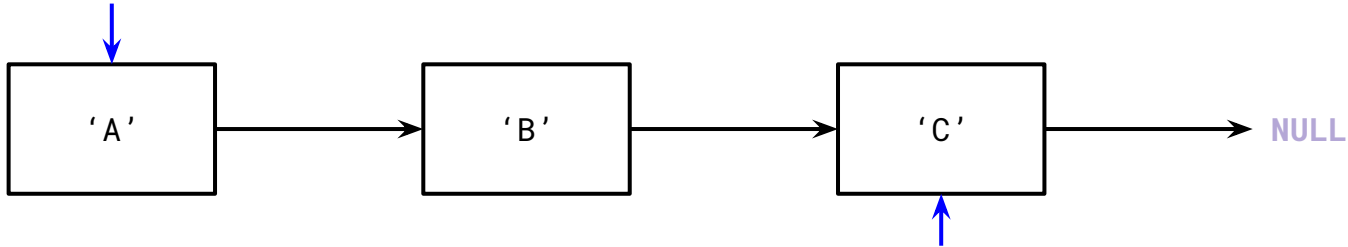
En definitiva, una pila puede ser una lista enlazada en la que los elementos se modifican únicamente mediante las operaciones:

- Insertar al principio - `push()`
- Elimino al principio - `pop()`



Implementaciones - Colas (Queue)

```
node_t* tail = &firstNode;
```



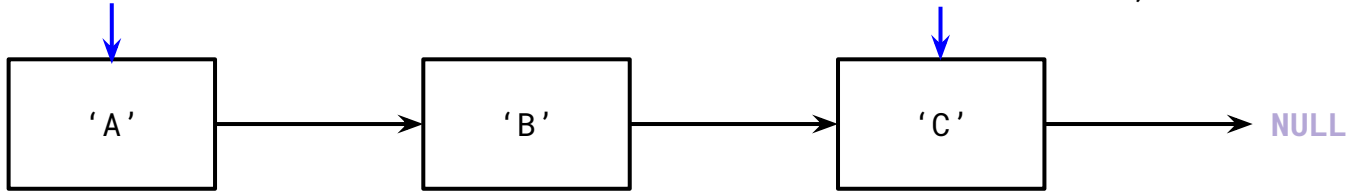
```
node_t* head = &lastNode;
```



Implementaciones - Colas (Queue)

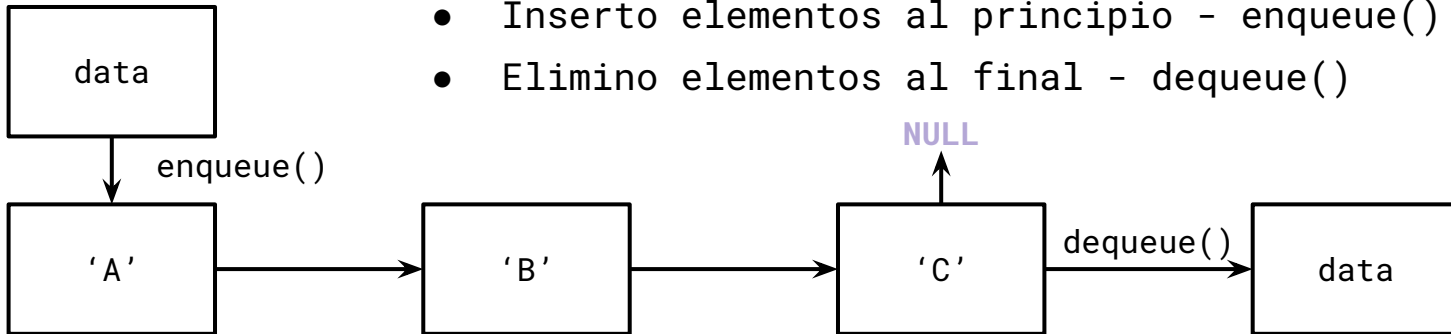
```
node_t* tail = &firstNode;
```

```
node_t* head = &lastNode;
```




De manera similar, una cola puede ser implementada con listas enlazadas si interactuamos con ella mediante las operaciones:

- Inserto elementos al principio - enqueue()
- Elimino elementos al final - dequeue()





Listas doblemente enlazadas



```
typedef struct{  
    int data;  
    struct dnode_t *next;  
    struct dnode_t *prev;  
}dnode_t;
```

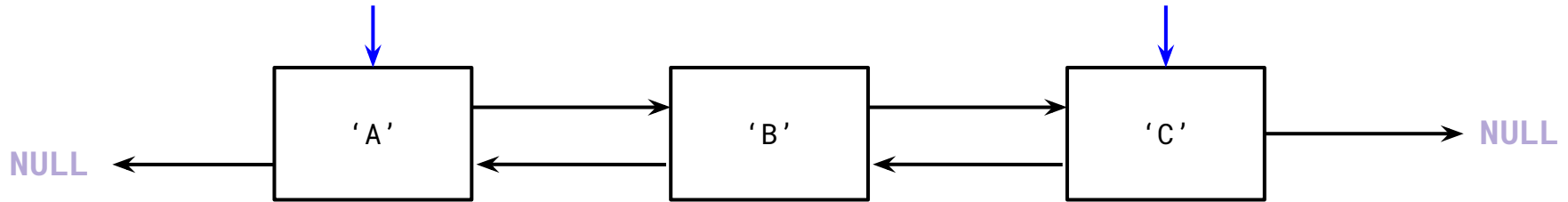


Listas doblemente enlazadas

```
typedef struct{  
    int data;  
    struct dnode_t *next;  
    struct dnode_t *prev;  
}dnode_t;
```

`dnode_t* head = &firstNode;`

`dnode_t* tail = &lastNode;`





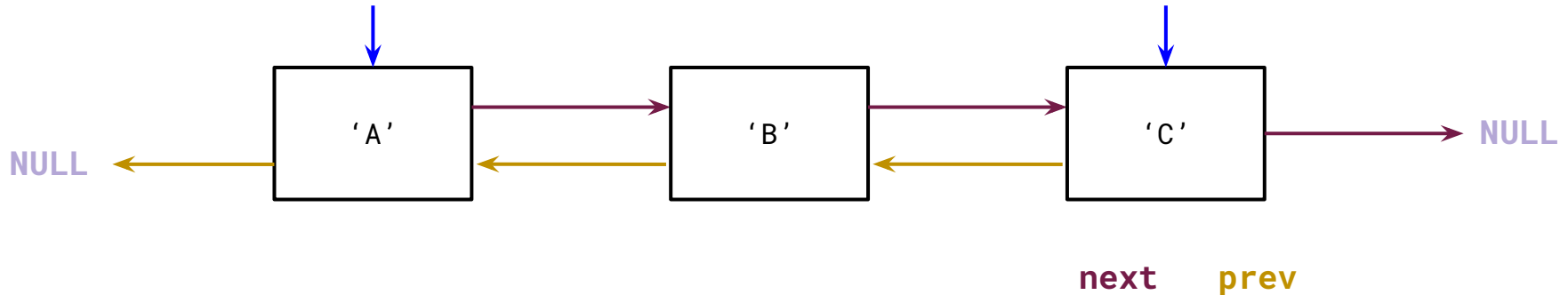
Listas doblemente enlazadas

```
typedef struct{  
    int data;  
    struct dnode_t *next;  
    struct dnode_t *prev;  
}dnode_t;
```

Funcionan igual que las listas enlazadas, pero los nodos están conectados en dos sentidos

```
dnode_t* head = &firstNode;
```

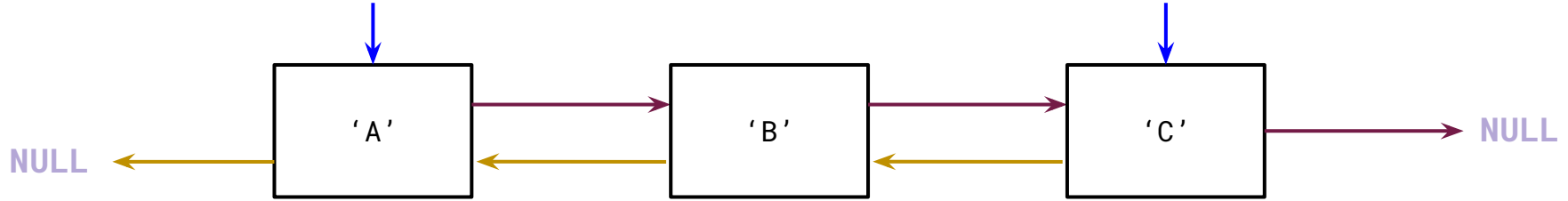
```
dnode_t* tail = &lastNode;
```



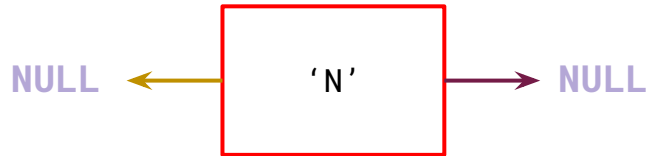
Listas dobles - Insertar al principio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



1. Creo nuevo nodo

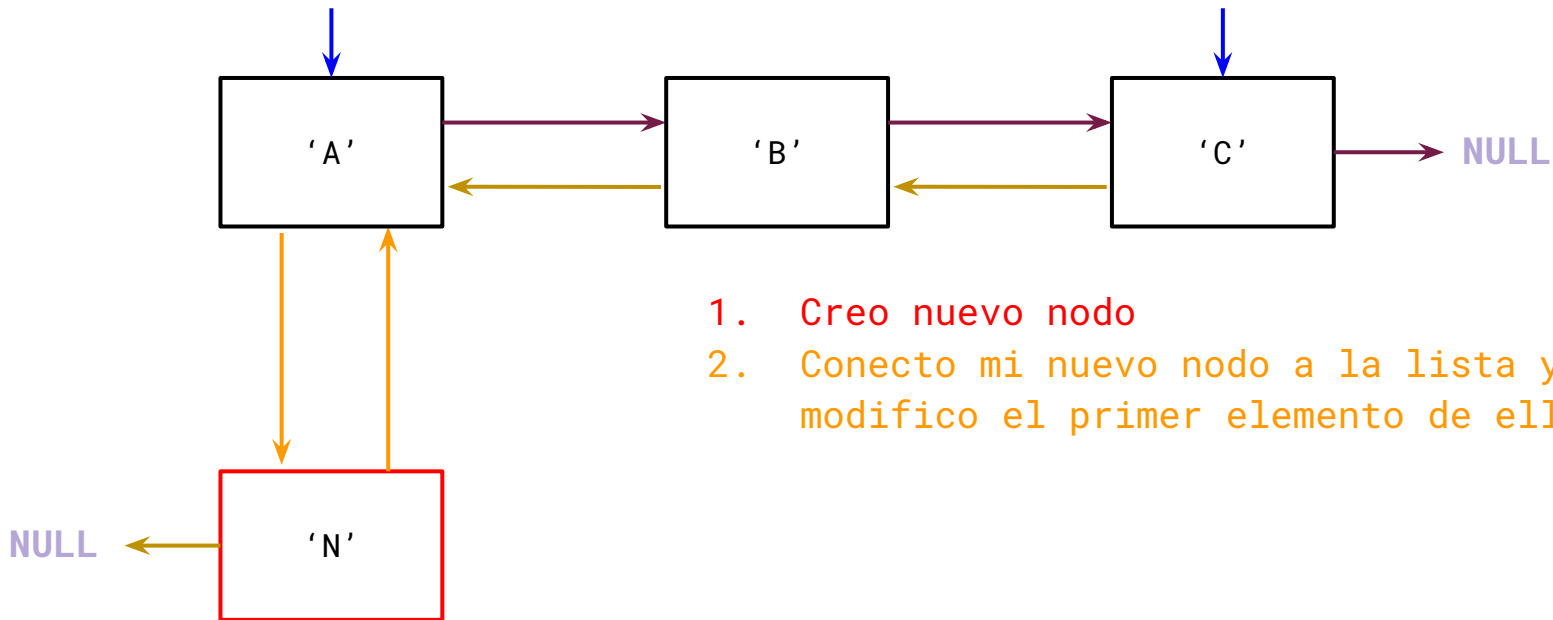




Listas dobles - Insertar al principio

```
dnode_t* head = &firstNode;
```

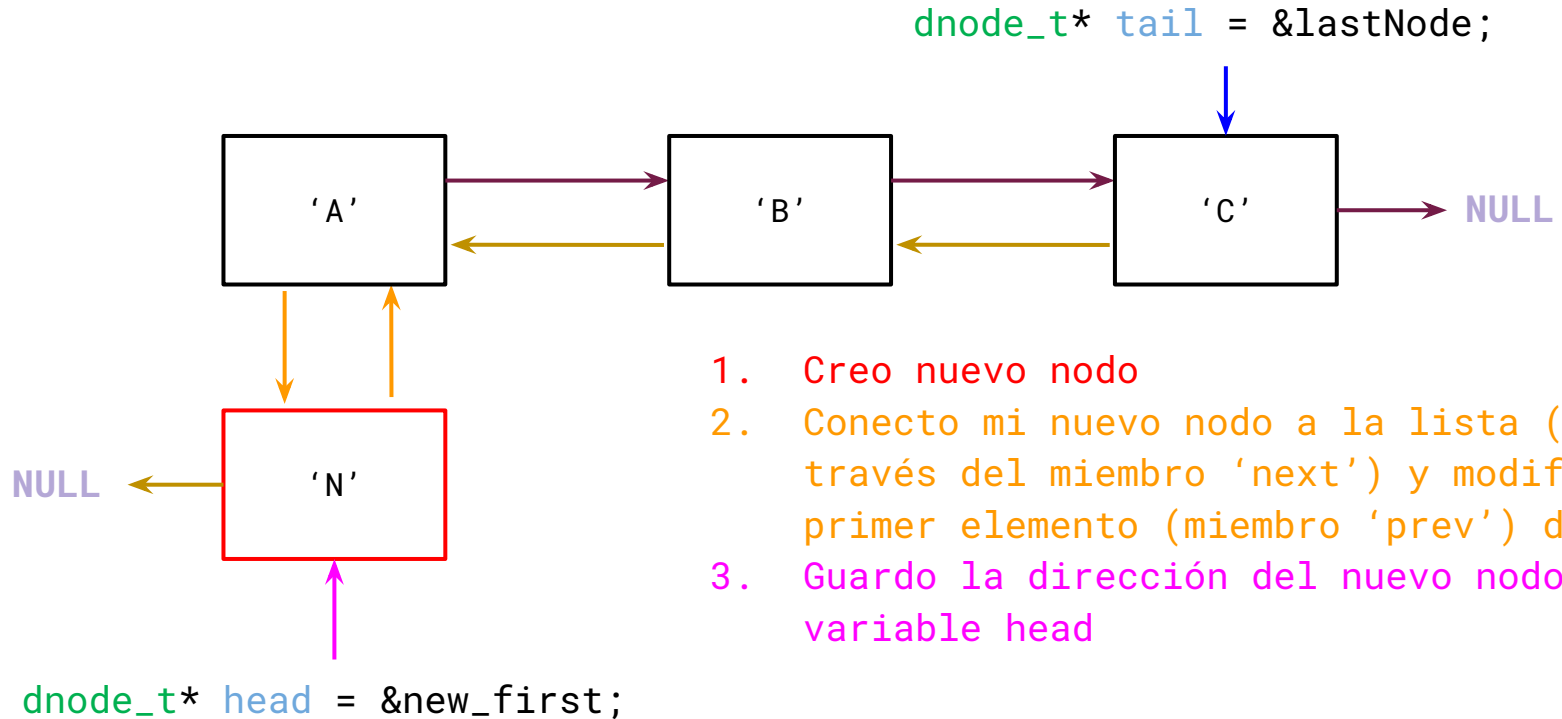
```
dnode_t* tail = &lastNode;
```



1. Creo nuevo nodo
2. Conecto mi nuevo nodo a la lista y modifico el primer elemento de ella



Listas dobles - Insertar al principio

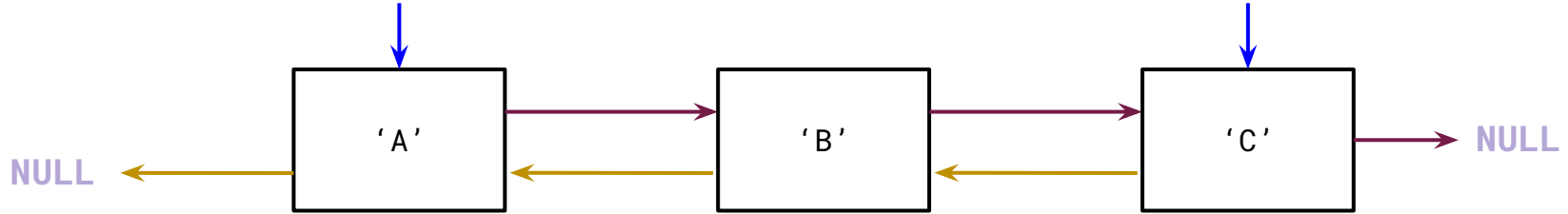




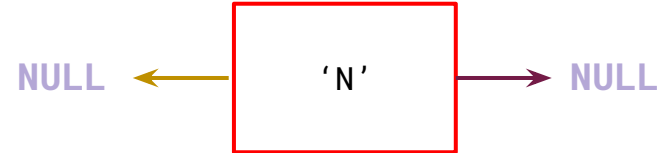
Listas dobles - Insertar al final

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



1. Creo nuevo nodo

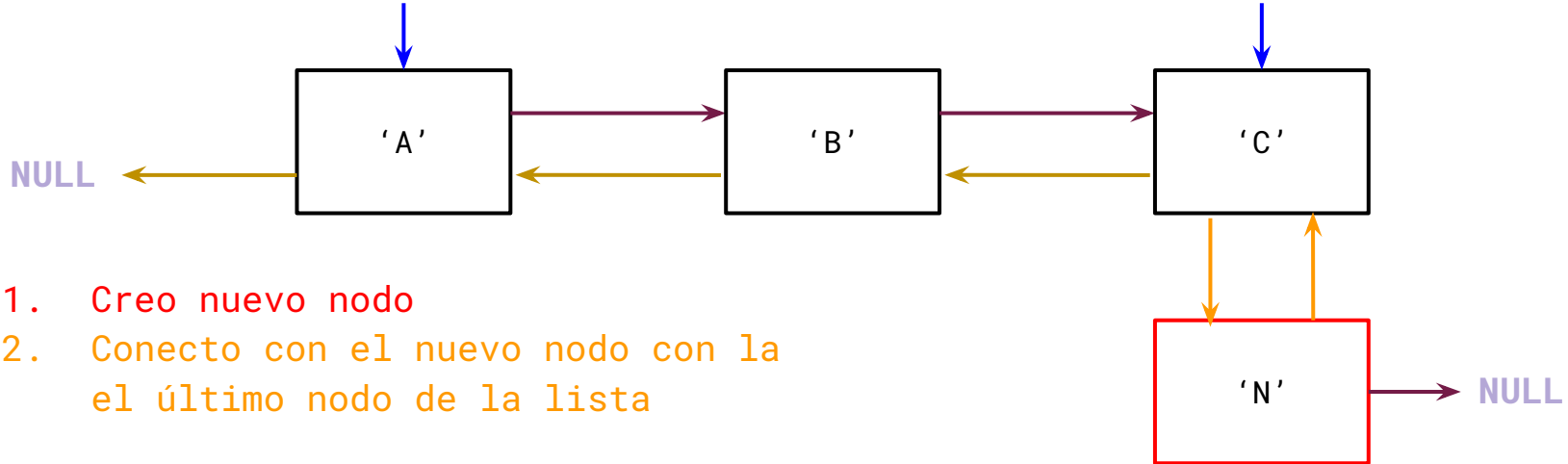




Listas dobles - Insertar al final

```
dnode_t* head = &firstNode;
```

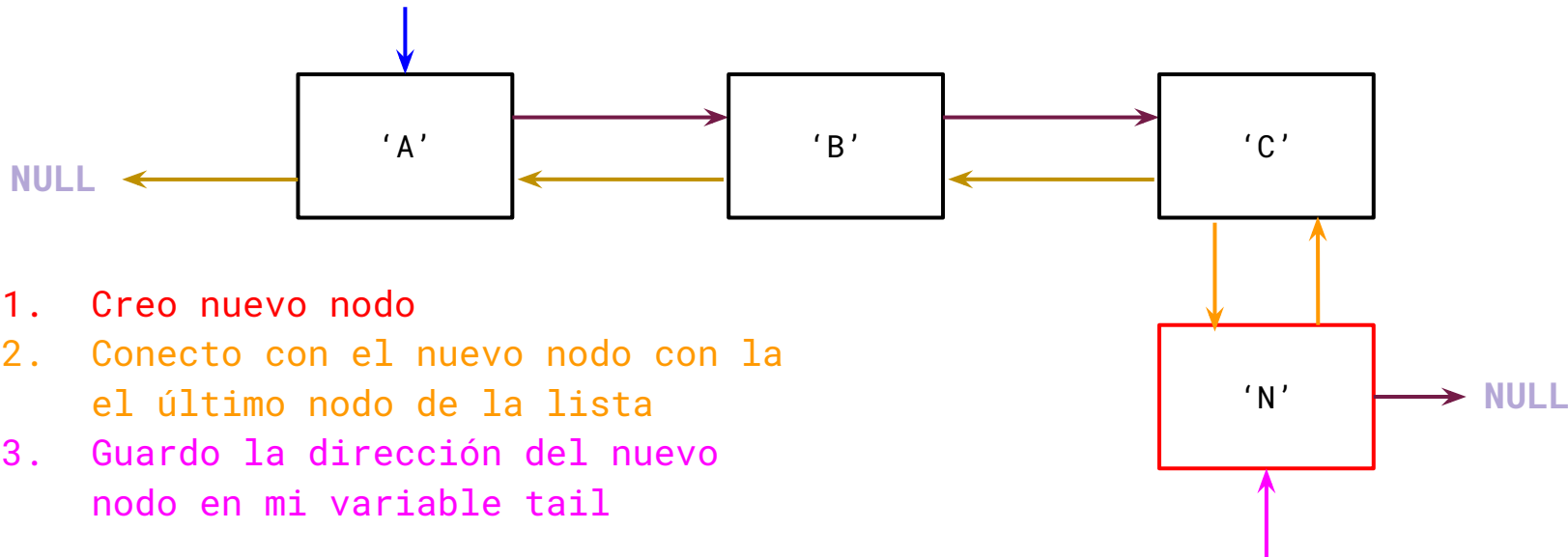
```
dnode_t* tail = &lastNode;
```





Listas dobles - Insertar al final

```
dnode_t* head = &firstNode;
```



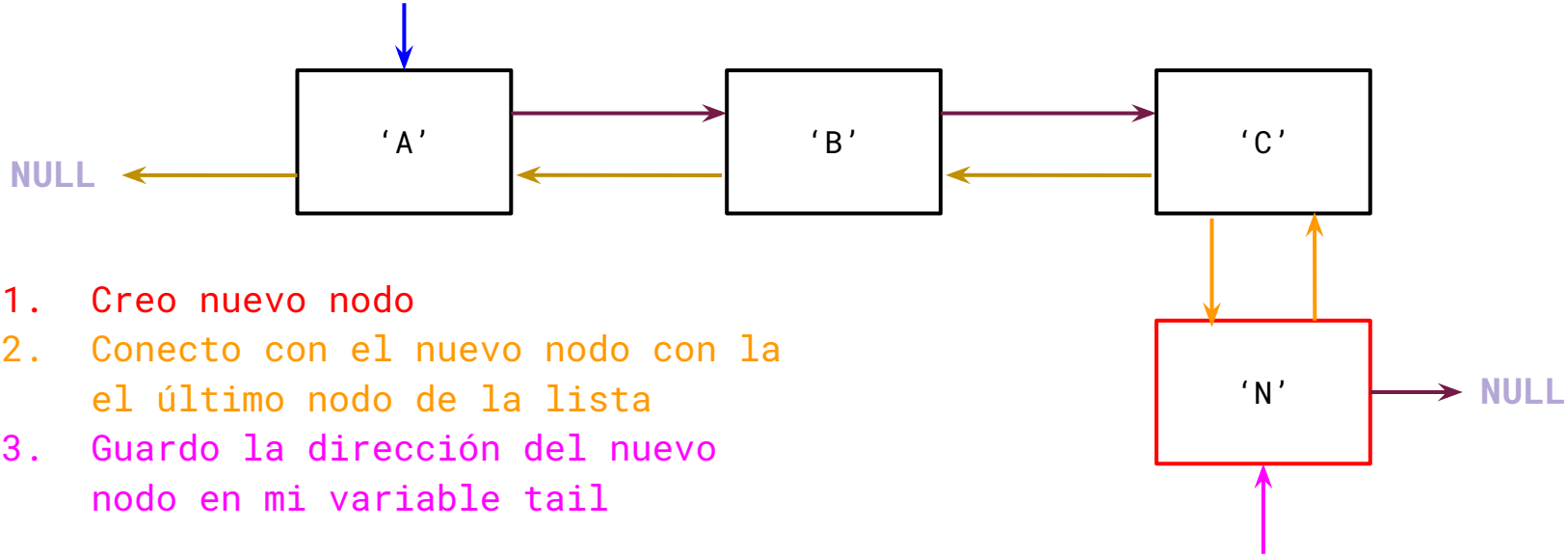
1. Creo nuevo nodo
2. Conecto con el nuevo nodo con la el último nodo de la lista
3. Guardo la dirección del nuevo nodo en mi variable tail

```
dnode_t* tail = &lastNode;
```



Listas dobles - Insertar al final

```
dnode_t* head = &firstNode;
```



1. Creo nuevo nodo
2. Conecto con el nuevo nodo con la el último nodo de la lista
3. Guardo la dirección del nuevo nodo en mi variable tail

obs. A diferencia de la listas simples, no necesito recorrer hasta el final, porque ya se donde termina la lista (tail)

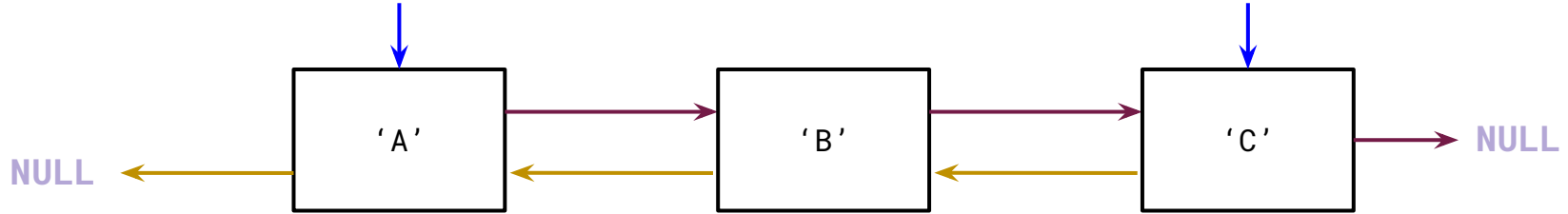
```
dnode_t* tail = &lastNode;
```



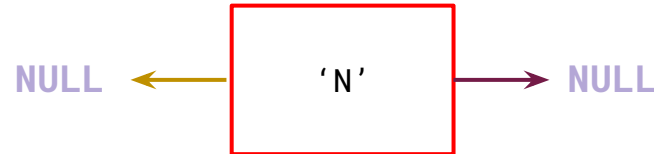
Listas dobles - Insertar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



1. Creo nuevo nodo

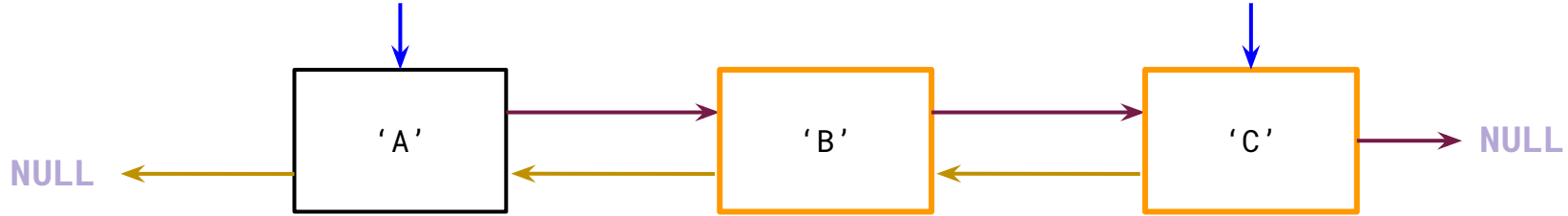




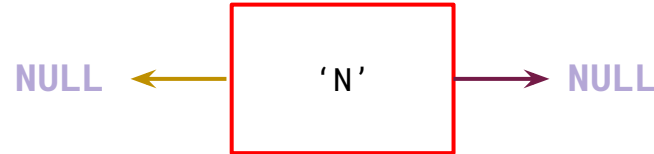
Listas dobles - Insertar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



1. Creo nuevo nodo
2. Me muevo a donde quiero insertar (desde cualquier extremo)

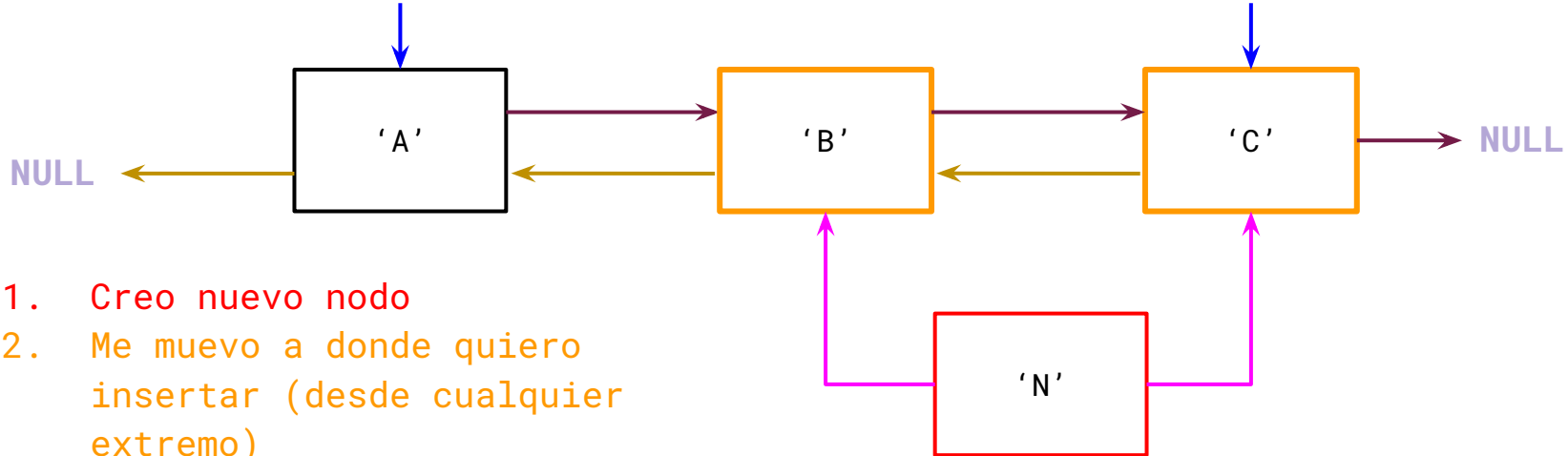




Listas dobles - Insertar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



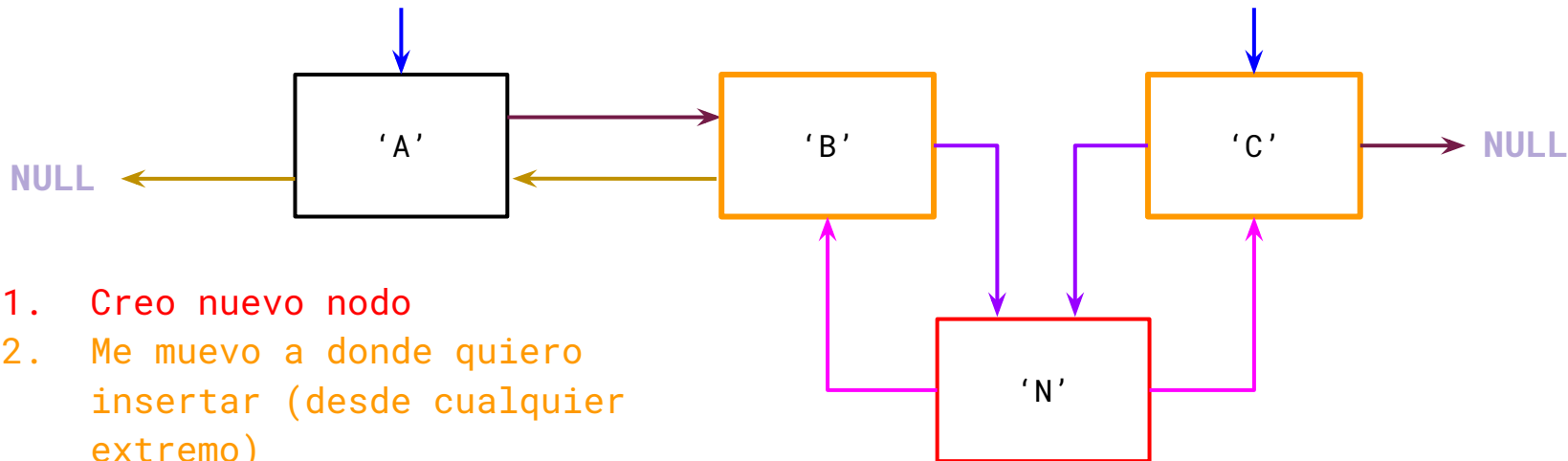
1. Creo nuevo nodo
2. Me muevo a donde quiero insertar (desde cualquier extremo)
3. Conecto mi nuevo nodo



Listas dobles - Insertar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



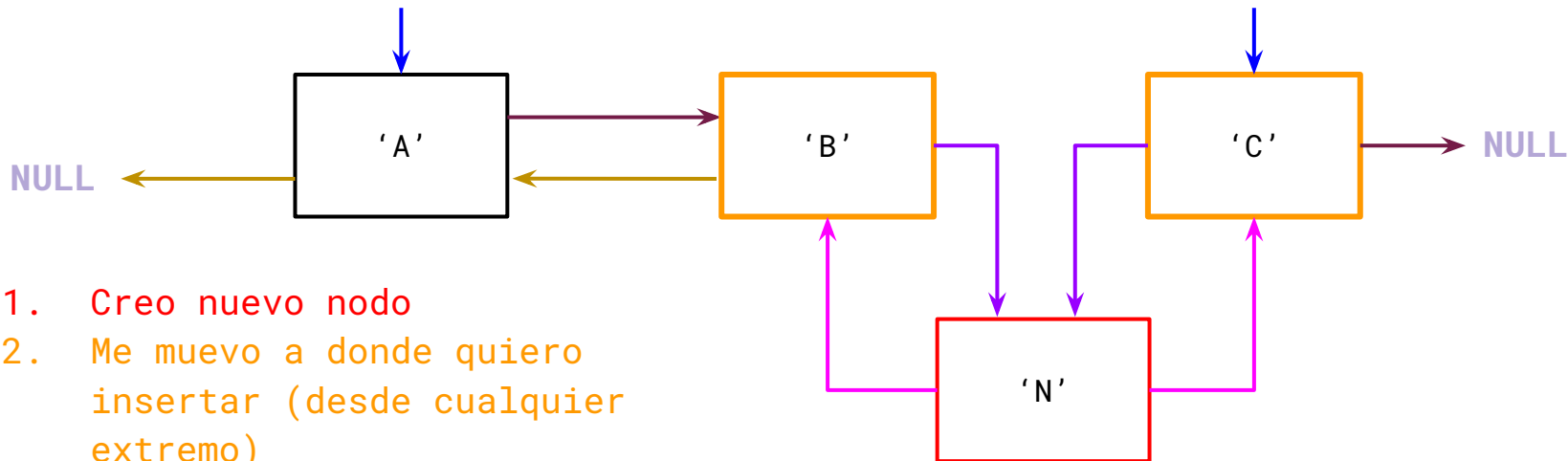
1. Creo nuevo nodo
2. Me muevo a donde quiero insertar (desde cualquier extremo)
3. Conecto mi nuevo nodo
4. Modifico los adyacentes



Listas dobles - Insertar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



1. Creo nuevo nodo
2. Me muevo a donde quiero insertar (desde cualquier extremo)
3. Conecto mi nuevo nodo
4. Modifico los adyacentes

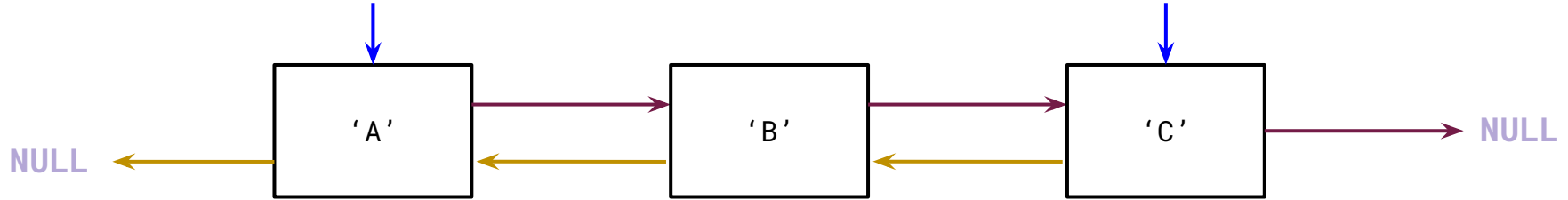
obs. Siempre es buena práctica primero conectar el nodo nuevo, y luego modificar la lista pre-existente.



Listas dobles - Eliminar al principio

```
dnode_t* head = &firstNode;
```

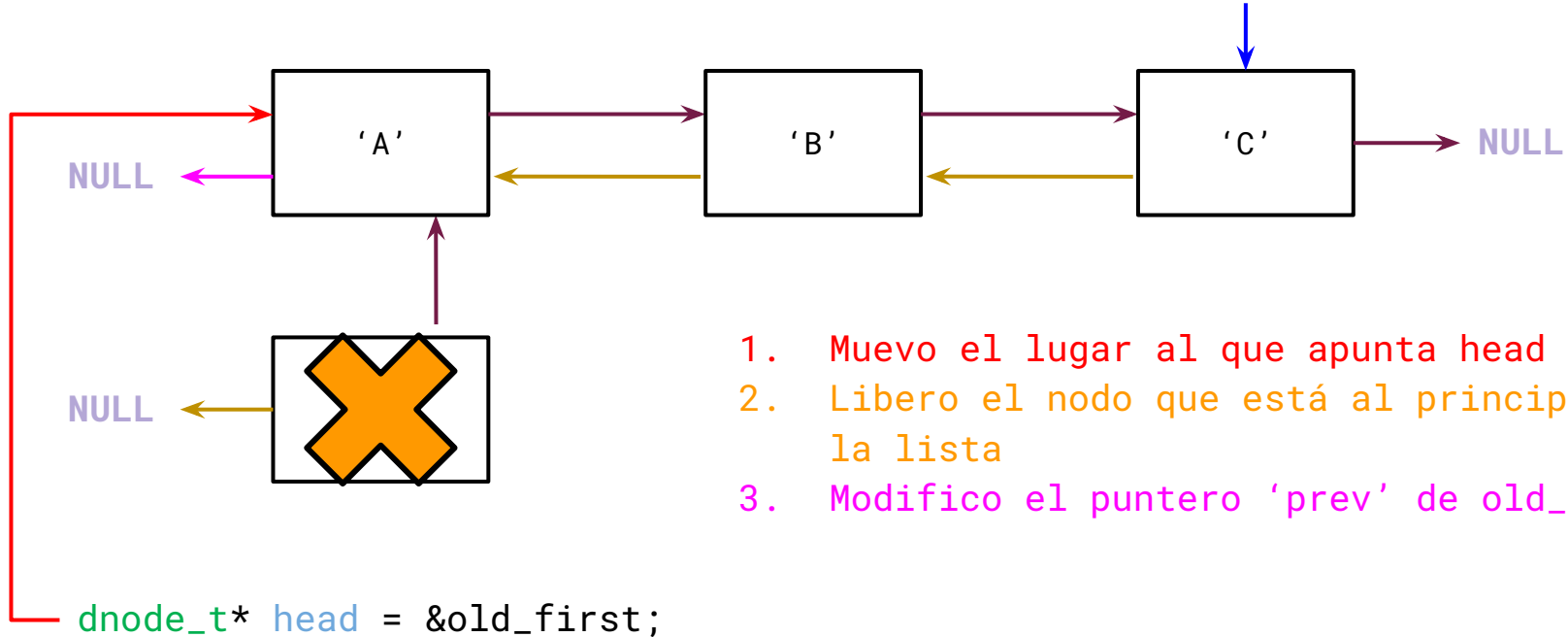
```
dnode_t* tail = &lastNode;
```





Listas dobles - Eliminar al principio

```
dnode_t* tail = &lastNode;
```



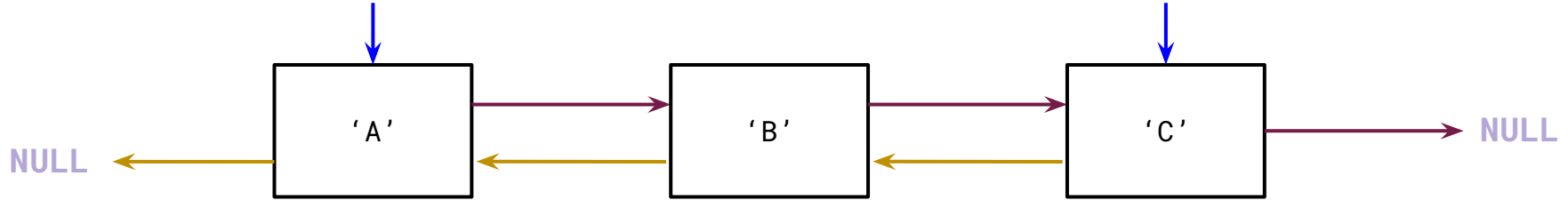
1. Muevo el lugar al que apunta head
2. Libero el nodo que está al principio de la lista
3. Modifico el puntero 'prev' de old_first



Listas dobles - Eliminar al final

```
dnode_t* head = &firstNode;
```

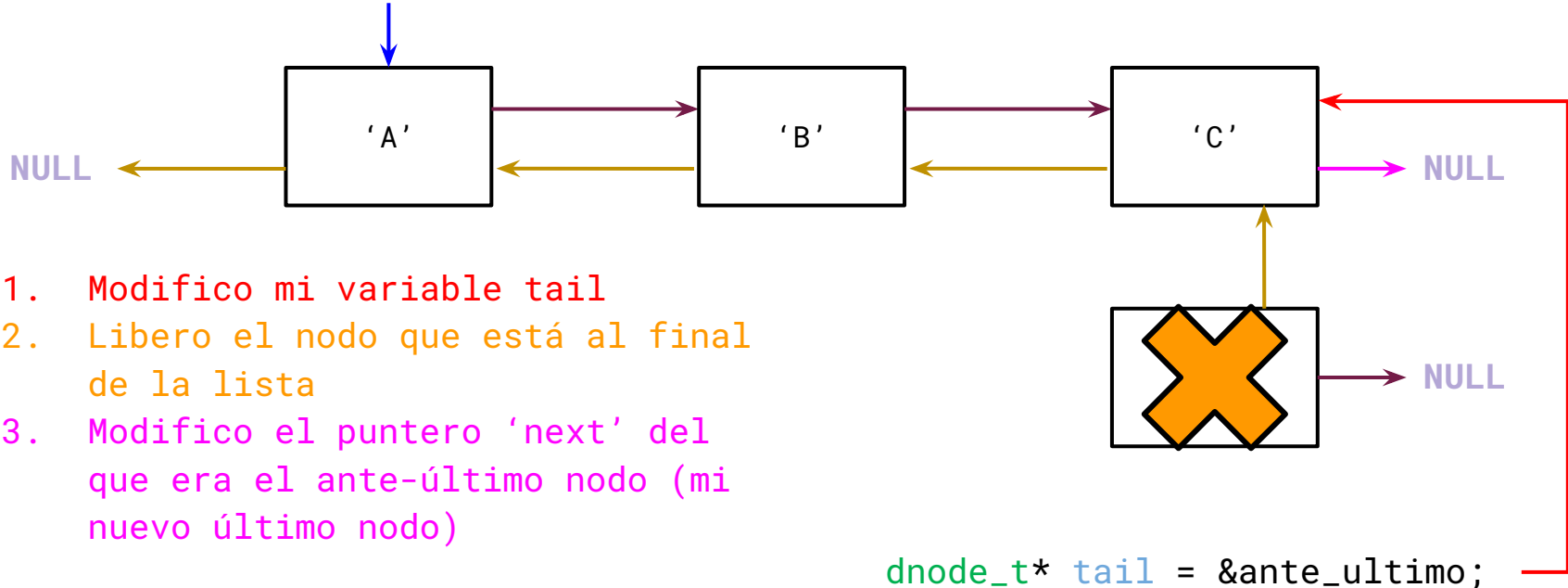
```
dnode_t* tail = &lastNode;
```





Listas dobles - Eliminar al final

```
dnode_t* head = &firstNode;
```



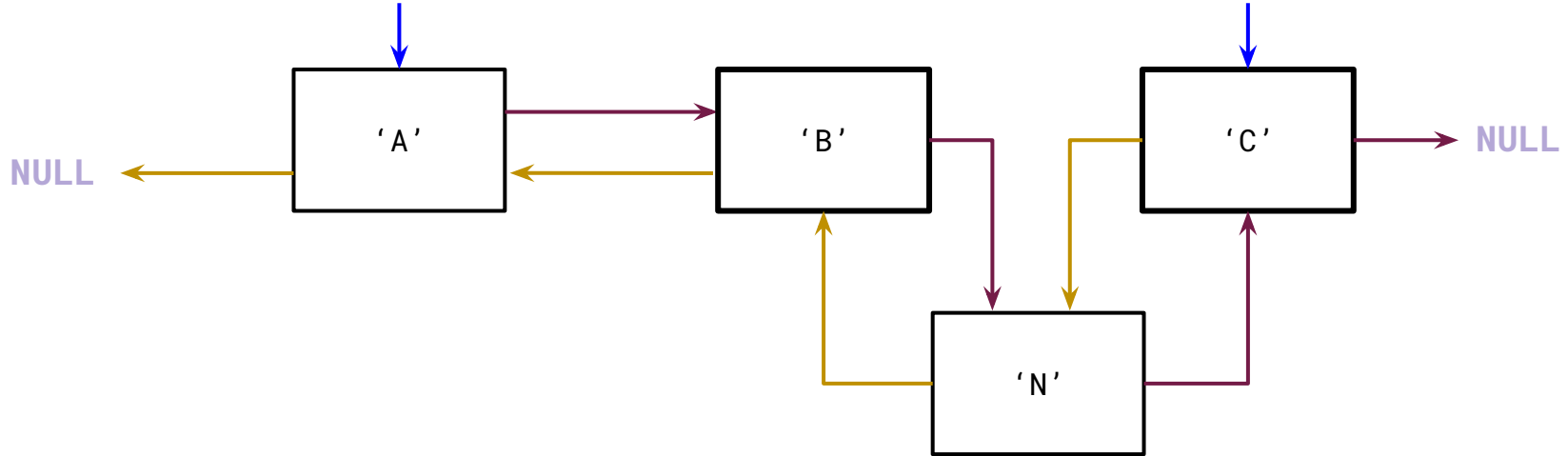
1. Modifico mi variable tail
2. Libero el nodo que está al final de la lista
3. Modifico el puntero 'next' del que era el ante-último nodo (mi nuevo último nodo)



Listas dobles - Eliminar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```

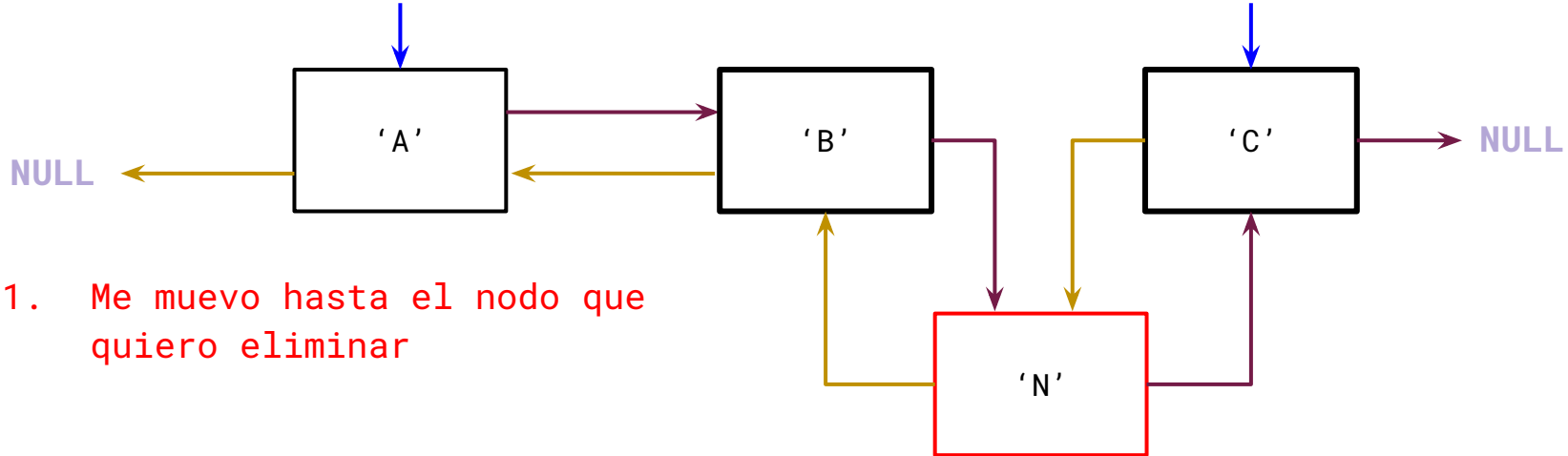




Listas dobles - Eliminar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```



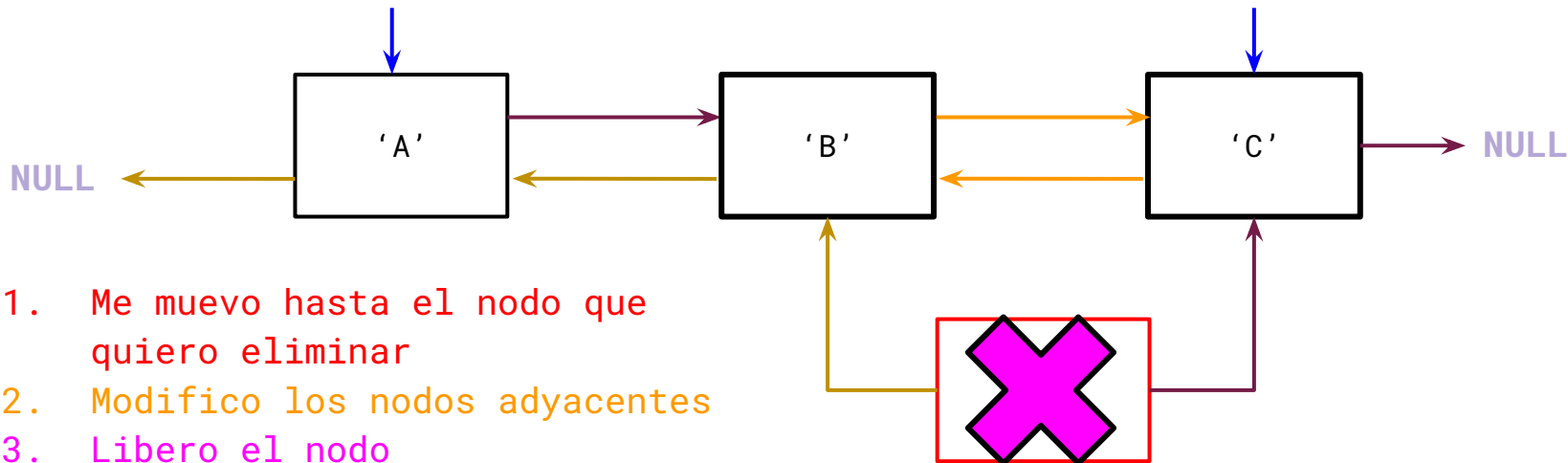
1. Me muevo hasta el nodo que quiero eliminar



Listas dobles - Eliminar en el medio

```
dnode_t* head = &firstNode;
```

```
dnode_t* tail = &lastNode;
```





Arrays - Insertar en el medio

0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0	10	20	30	40	50

Supongamos que quiero insertar el 25, manteniendo la lista ordenada



Arrays - Insertar en el medio

0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
0	10	20	30	40	50	?
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]

Supongamos que quiero insertar el 25, manteniendo la lista ordenada

1. Cambio el tamaño del array (realloc)



Arrays - Insertar en el medio

0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
0	10	20	30	30	40	50
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]

Supongamos que quiero insertar el 25, manteniendo la lista ordenada

1. Cambio el tamaño del array (realloc)
2. Muevo los valores ya existentes una posición a la derecha (memmove)



Arrays - Insertar en el medio

0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
0	10	20	25	30	40	50
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]

Supongamos que quiero insertar el 25, manteniendo la lista ordenada

1. Cambio el tamaño del array (realloc)
2. Muevo los valores ya existentes una posición a la derecha (memmove)
3. Escribo el valor a insertar en la posición correspondiente



Arrays - Insertar en el medio

0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
0	10	20	25	30	40	50
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]

Supongamos que quiero insertar el 25, manteniendo la lista ordenada

1. Cambio el tamaño del array (realloc)
2. Muevo los valores ya existentes una posición a la derecha (memmove)
3. Escribo el valor a insertar en la posición correspondiente

obs. Eliminar un elemento del medio es un proceso similar, pero achico el array en vez de agrandarlo.



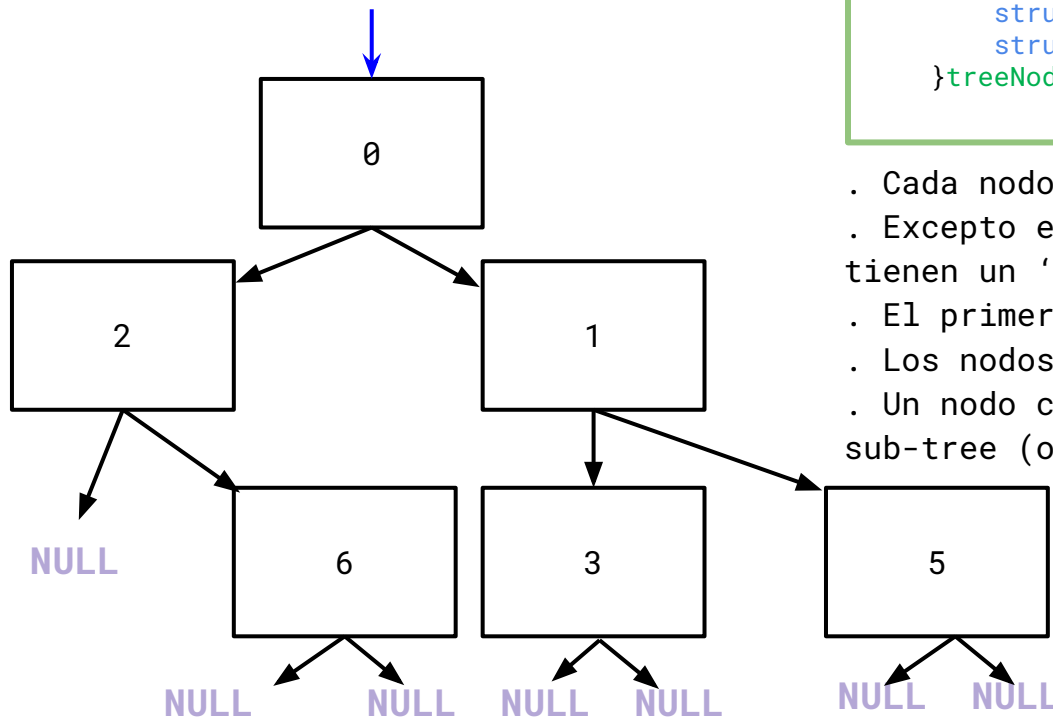
Arrays vs Linked Lists

	Arrays			Linked Lists		
	Worst	Average	Best	Worst	Average	Best
Acceso	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Inserción / Eliminación	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Memory Overhead	Si elijo bien el tamaño del array, no hay overhead			Necesito memoria adicional para los punteros		
Memory Efficiency	Puede utilizar memoria extra			Como cambia de tamaño dinámicamente, no utiliza memoria extra		

Estructuras más complejas - Árbol binario

```
dnode_t* root = &firstNode;
```

```
typedef struct{  
    int data;  
    struct treeNode_t *left_child;  
    struct treeNode_t *right_child;  
}treeNode_t;
```

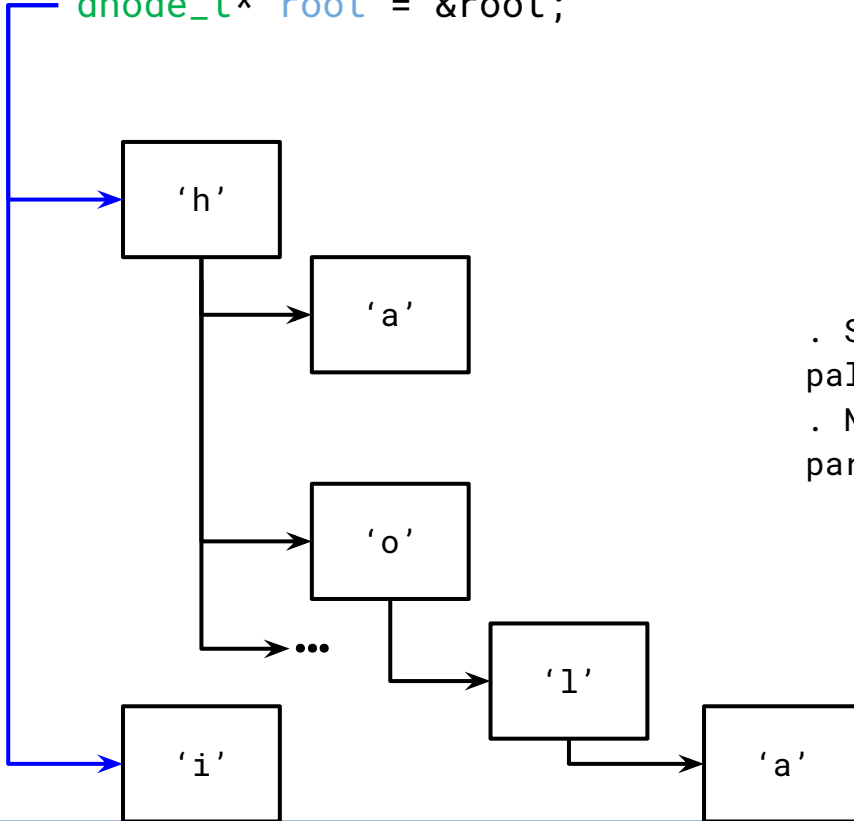


- . Cada nodo tiene 2 hijos.
- . Excepto el primer nodo, todos los nodos tienen un 'padre'
- . El primer nodo se llama 'root'
- . Los nodos sin hijos se llaman 'leafs' (hojas)
- . Un nodo con todos sus descendientes se llama sub-tree (o rama)



Estructuras más complejas - Trie

```
dnode_t* root = &root;
```



```
typedef struct{  
    struct trieNode_t *children[26];  
    int8_t isEndOfWord;  
}trieNode_t;
```

- . Se utilizan mucho para autocompletar palabras, corrección de ortografía, etc.
- . Notar que no se pusieron todas las flechitas para facilitar el entendimiento de la figura