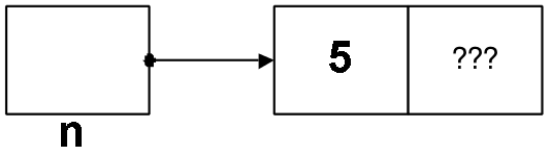
Listas simplemente enlazadas. Operaciones avanzadas

Insertar Ordenado (en una lista ordenada)

Deseamos insertar un nodo de acuerdo a un criterio de ordenamiento que vamos a fijar de antemano.
Los pasos a seguir serían:
Paso 1) Es siempre igual. Solicitar memoria para el nuevo nodo usando el puntero n.

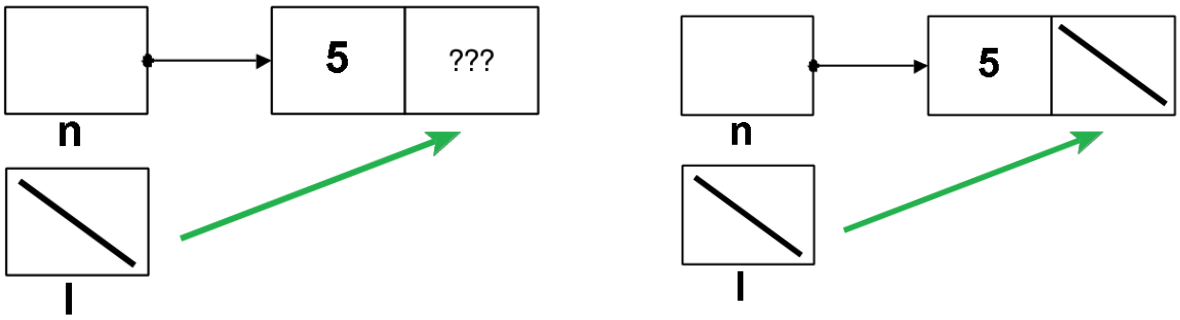


Para los siguientes pasos procedemos a evaluar 2 casos posibles:

- Caso 1) Insertar al principio
- Caso 2) Insertar en el medio o final de la lista (requiere búsqueda)

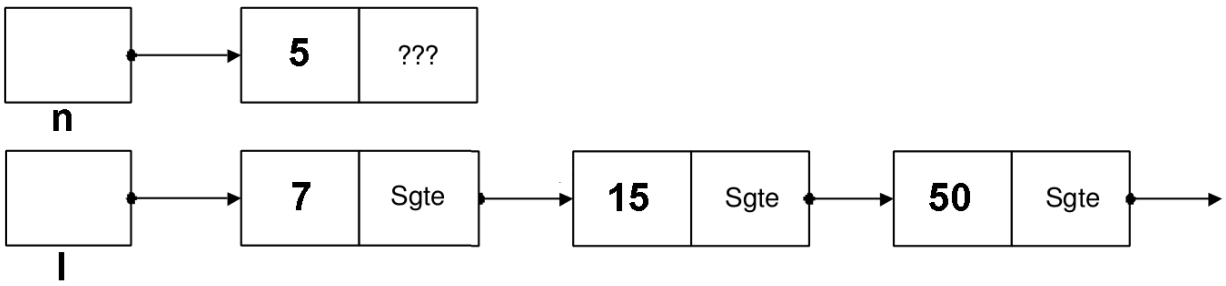
Caso 1.1) Lista vacía

Paso 2) Asignamos al campo Sgte del nodo apuntado por n (nuevo nodo) el valor de l (o sea NULL)

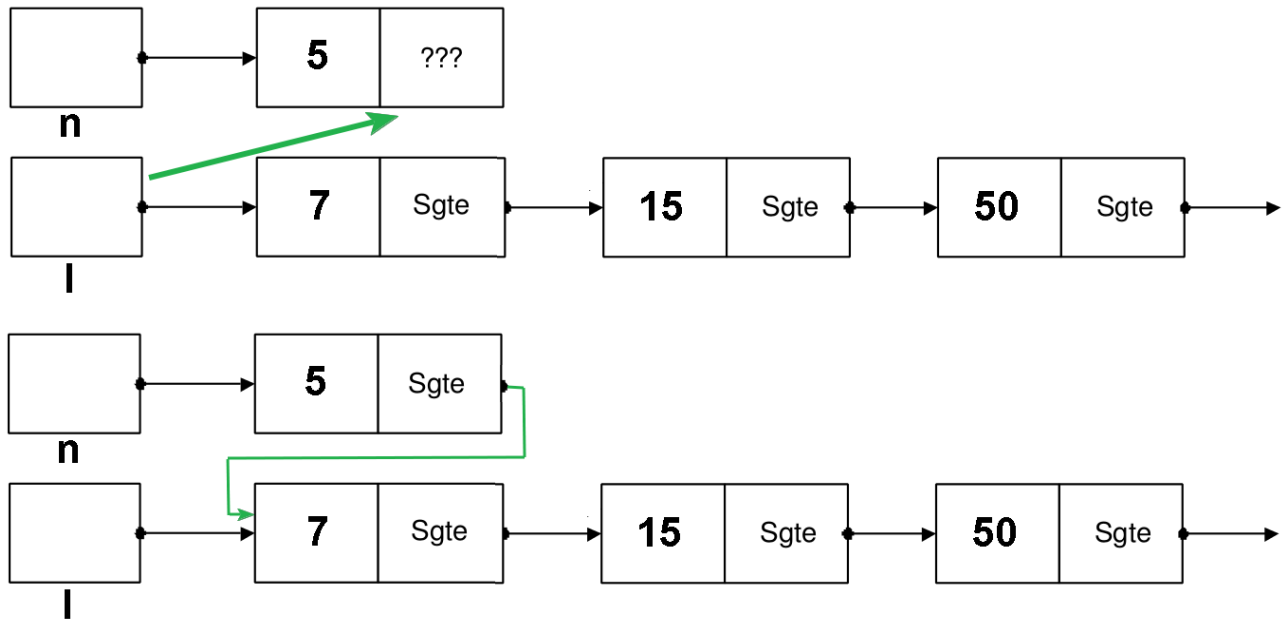


Paso 3) Retornamos n

Caso 1.2) Lista no vacía



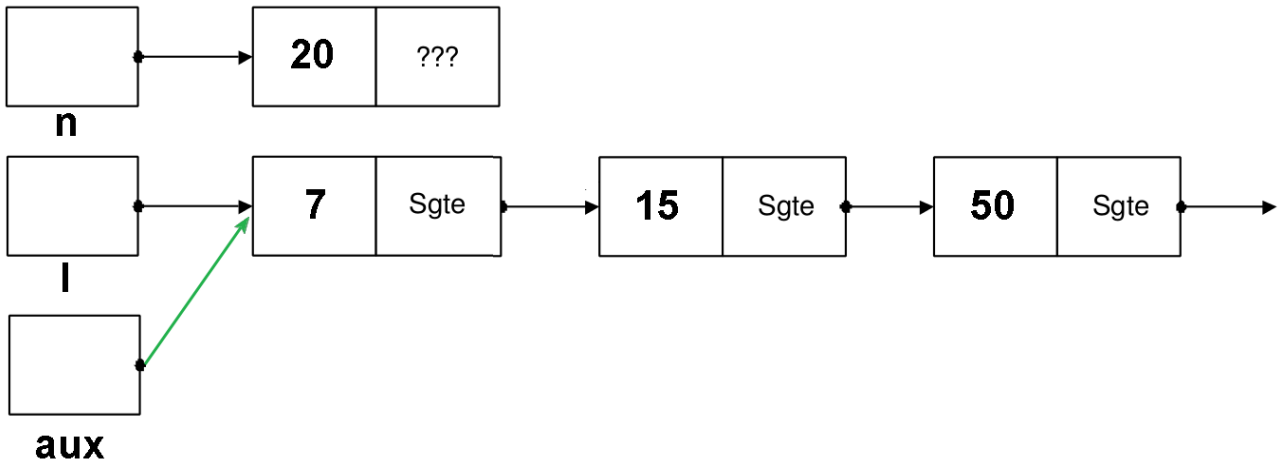
Paso 2) Asignamos al campo Sgte del nodo apuntado por n el valor de l



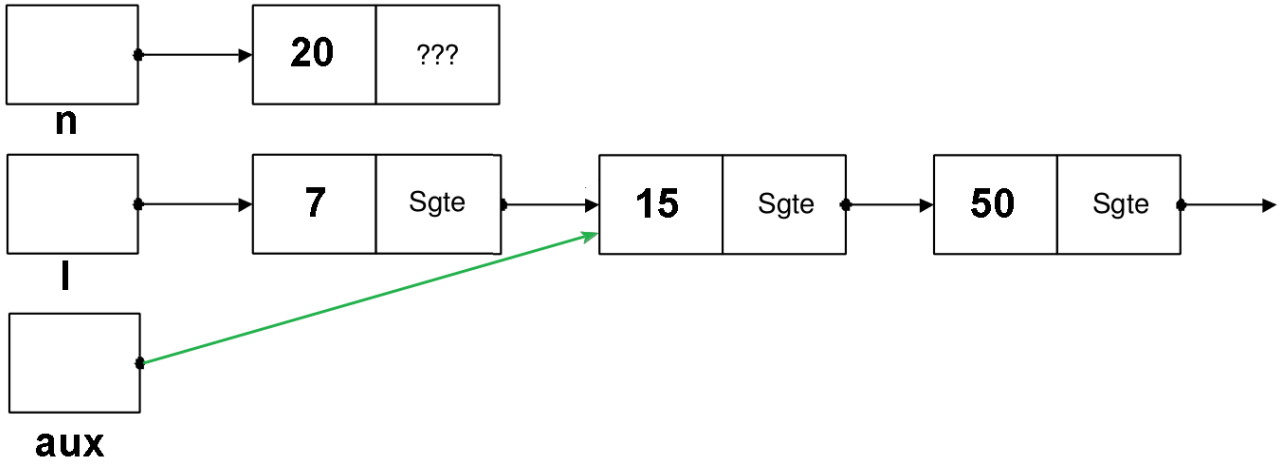
Paso 3) Retornamos n
Como vemos es procedimiento fue idéntico si la estaba o no vacía (casos 1.1 y 1.2).

Caso 2) Insertamos en el medio o final de la lista.
Tenemos la siguiente lista y deseamos insertar un nuevo elemento ordenado. Utilizamos un puntero auxiliar para recorrerla.

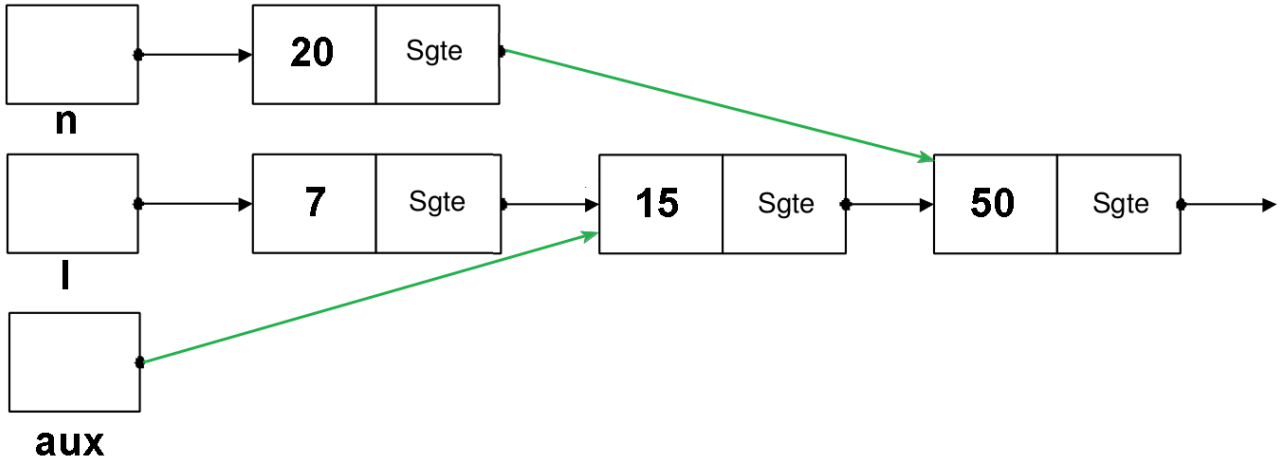
Paso 2) Creamos un puntero auxiliar que apunte al principio de la lista.



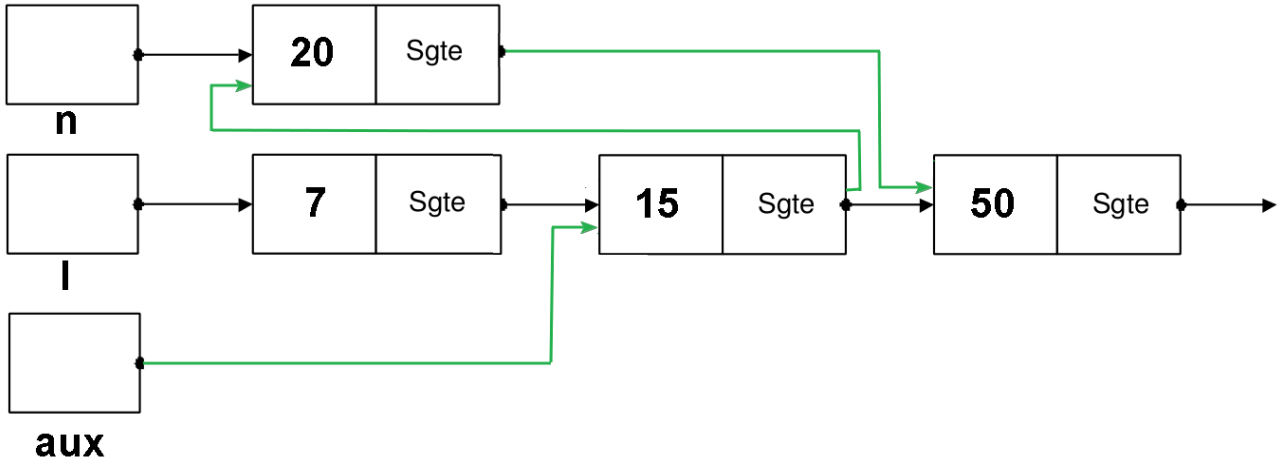
Paso 3) Procedemos a recorrer la lista buscando la posición correcta en la que se tiene que insertar el nodo. Preguntamos por el nodo Sgte para no perder la referencia (no pasarnos).
Importante: El primer nodo ya fue evaluado (por eso sabemos que estamos en el caso 2).



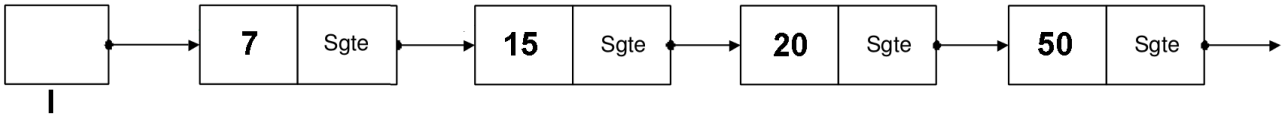
Paso 4) Al encontrar el nodo, el próximo paso es “enganchar” el nodo n que creamos en la posición que corresponde y esto se logra apuntando el campo Sgte del nodo n a lo que apunta la dirección Sgte del puntero aux.
Importante: aux apunta al nodo anterior a la posición a insertar.



Paso 5) Por último nos resta que el campo Sgte del nodo apuntado por aux (el anterior en orden) apunte al nodo recientemente creado.



Es importante seguir los pasos en ese orden para no perder referencias.
Finalmente nos queda la siguiente lista:



Paso 6) Retornarnos l

Para el caso en que el nodo a insertar es el último, el procedimiento es el mismo, solamente que el ciclo de búsqueda de posición correcta (paso 3) finaliza debido al fin de la lista (el puntero Sgte es NULL).

Función que realiza lo explicado:

```
/*Inserta en orden*/
nodo *insertar_ordenado(nodo *l, int d) {
    nodo *nuevo, *aux;

    nuevo = (nodo*) malloc(sizeof(nodo));
    nuevo->dato = d;
    if (l == NULL || l->dato > d) {
        /*Lista vacia o primer elemento > nuevo*/
        nuevo->sig = l;
        return nuevo;
    }
    /*Si llega hasta aca => lista no vacia y nuevo > primer*/
    aux = l;
    while (aux->sig != NULL && aux->sig->dato <= d)
        aux = aux->sig;
    /*Salio del ciclo => aux->sig es NULL o aux->sig->dato > d*/
    /*Inserto el nuevo entre aux y aux->sig*/
    nuevo->sig = aux->sig;
    aux->sig = nuevo;
    return l;
}
```

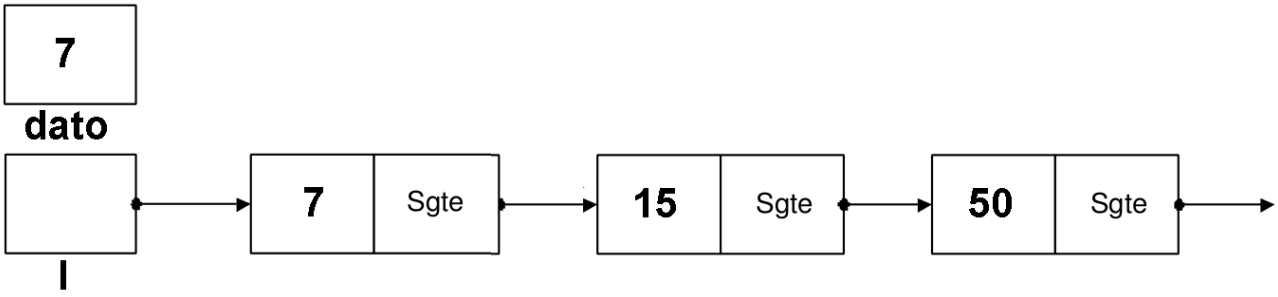
Eliminar Nodo

Para suprimir un nodo es necesario "desprenderlo" de la lista y luego ejecutar un **free()** sobre él. Esto se logra con un puntero que apunte al nodo.

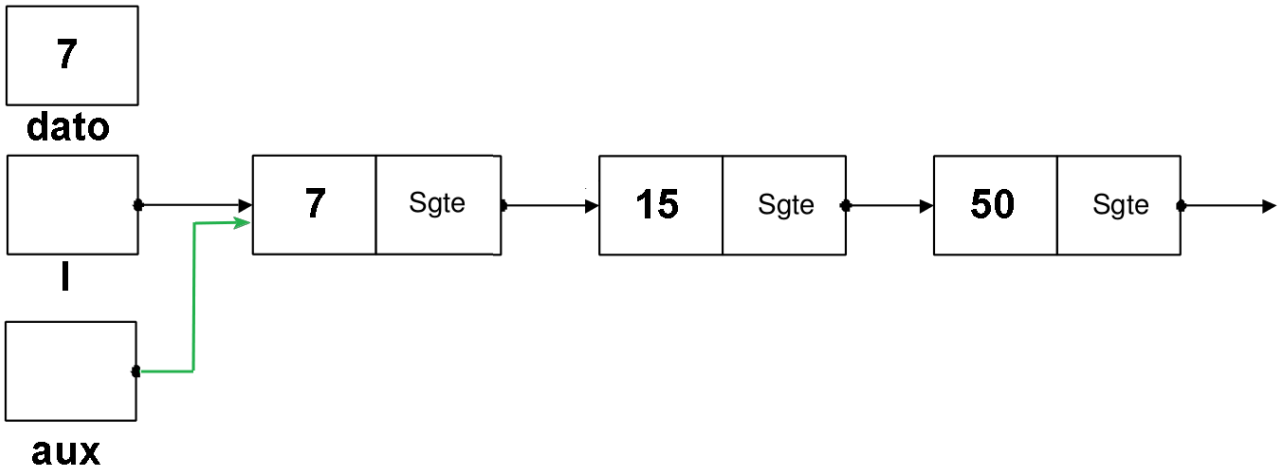
Alternativas:

- **Caso 1) Trivial: Si la lista está vacía no hacemos nada. Recibimos NULL y retornamos NULL.**
- Caso 2) El nodo a eliminar es el primero.
- Caso 3) El nodo a eliminar está en el medio o al final (requiere búsqueda)

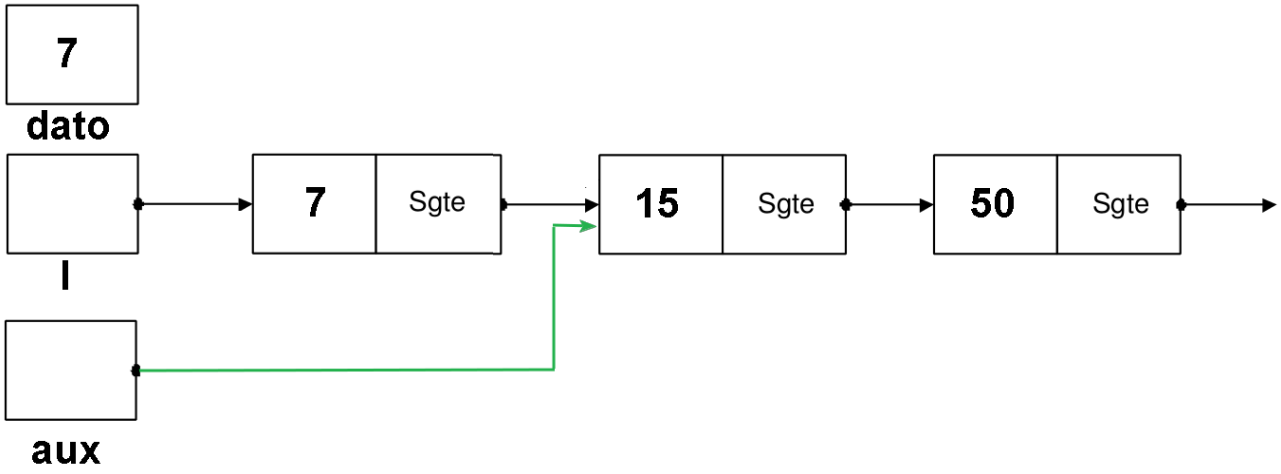
Caso 2) Tenemos la siguiente lista y queremos eliminar el nodo que contenga el dato 7.



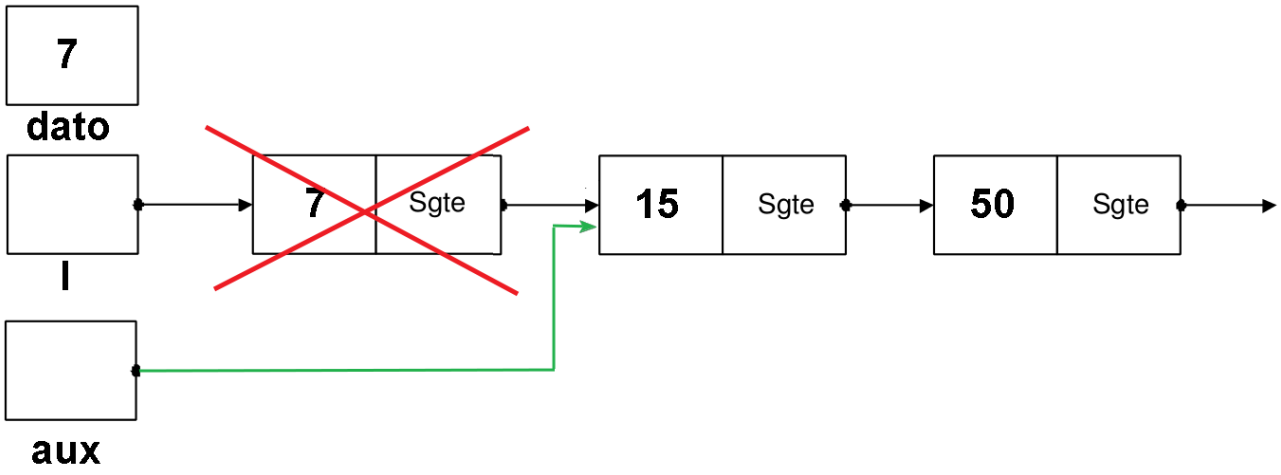
Paso 1) Utilizamos un puntero auxiliar para apuntar al primer elemento de la lista.



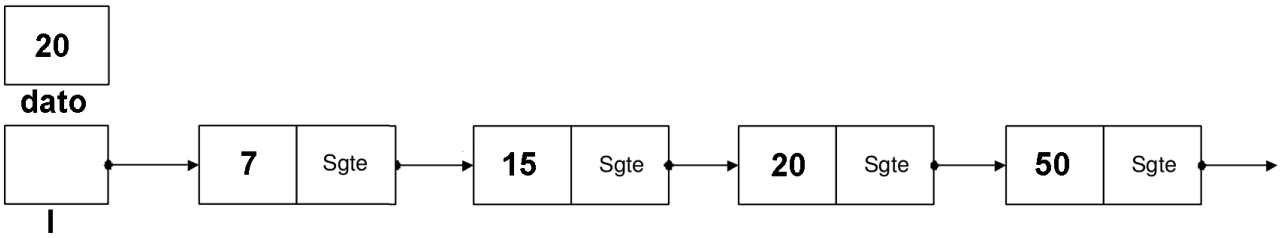
Paso 2) Apuntamos al segundo elemento de la lista. Esto lo logramos copiando en aux la dirección del puntero Sgte del nodo apuntado por l.



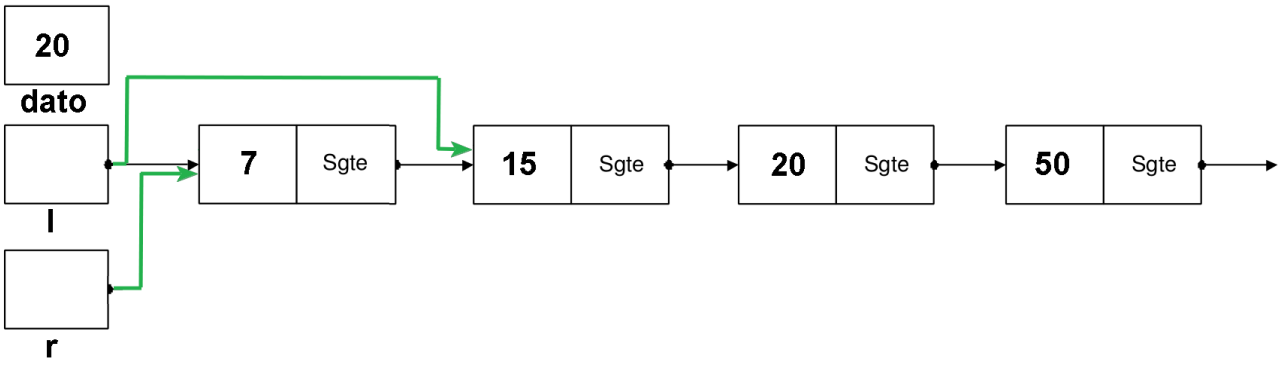
Paso 3) Eliminamos el nodo apuntado por l.



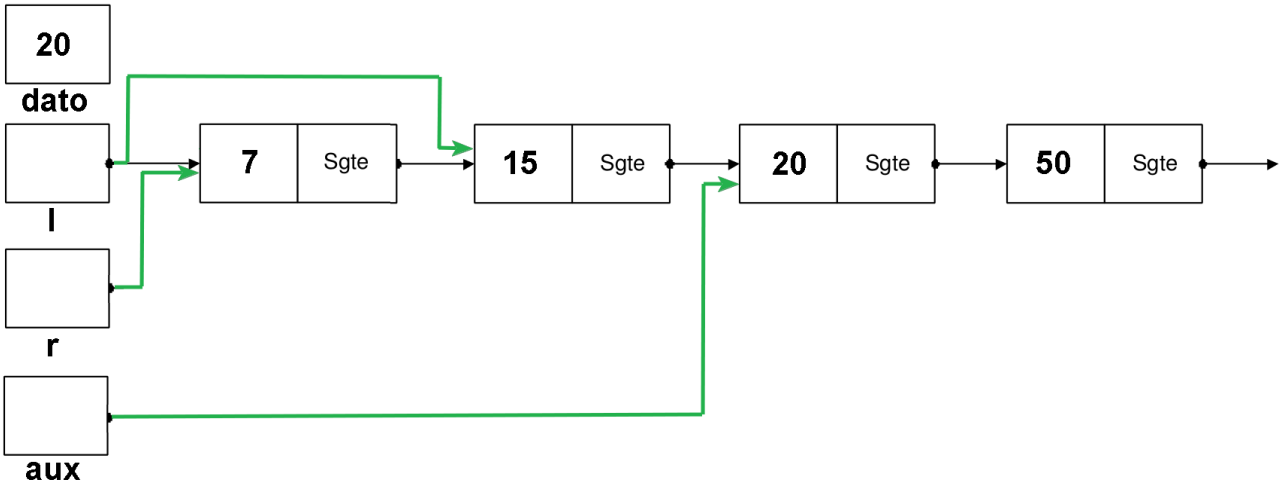
Caso 3) Disponemos de la siguiente lista. El nodo a eliminar es aquel que contenga en su interior el dato 20.



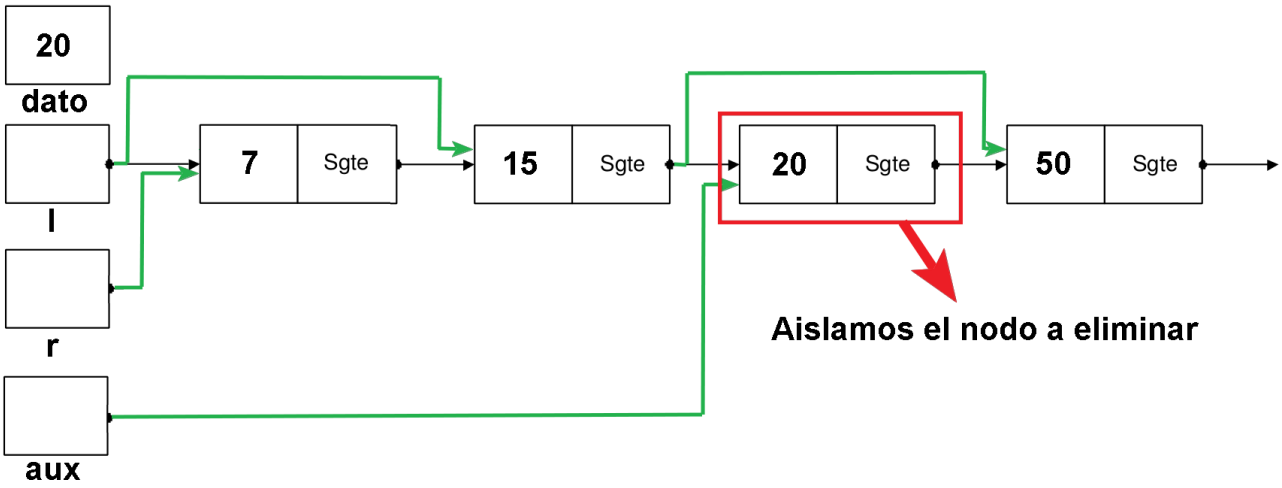
Paso 1) Guardamos en un puntero auxiliar la dirección al primer elemento de la lista
Paso 2) Avanzamos sobre l hasta encontrar el nodo a eliminar (nos posicionamos siempre sobre el nodo anterior al que se quiere eliminar)



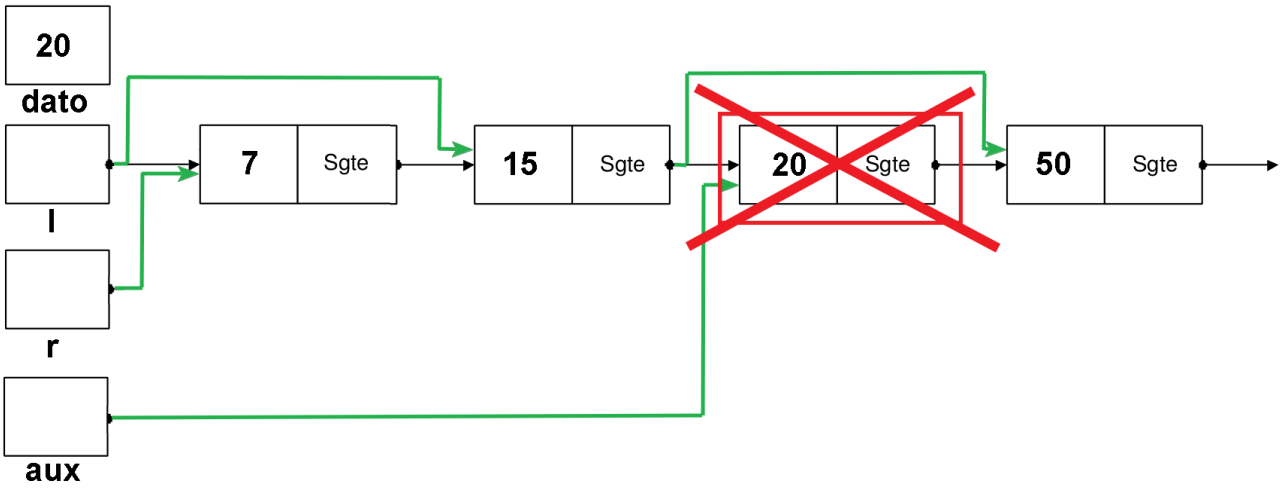
Paso 3) Apuntamos al nodo a eliminar. Guardamos en el puntero aux la dirección contenida en el puntero Sgte del nodo apuntado por l.



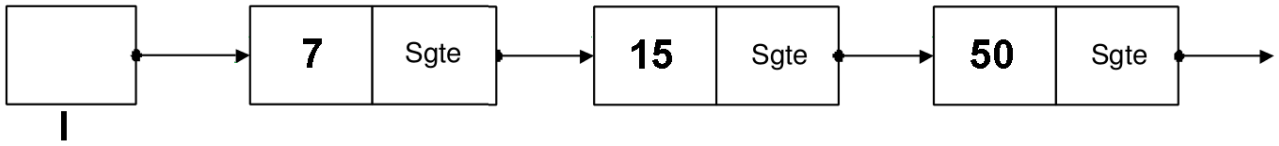
Paso 4) Establecemos un puente entre el nodo que contiene el dato 15 y el nodo que contiene el dato 50 para aislar el nodo a eliminar sin perder ninguna referencia. El puntero Sgte del nodo apuntado por l pasa a apuntar a la dirección contenida en el puntero Sgte del nodo apuntado por aux.



Paso 5) Liberamos la memoria del nodo apuntado por aux.



Finalmente nos queda la siguiente lista:



Paso 6) Retornamos r.

Función que realiza lo explicado:

```
/* Elimina un nodo de la lista */
nodo *eliminar (nodo* l, int d) {
    nodo *ret = l, *aux;

    if (l == NULL) { /* 1er caso: Si la lista es nula no hay que hacer nada */
        printf ("La lista esta vacia\n");
    }
    else { /* La no lista es nula */
        if (l->dato == d){ /* 2do caso: el nodo a borrar es el primero */
            ret = l->sig;
            free (l);
            printf ("Elemento eliminado\n");
        }
        else { /* 3er caso: el nodo a borrar no es el primero */
            while (l->sig != NULL && l->sig->dato != d)
                l = l->sig;
            if (l->sig != NULL) { /* Si l->sig es NULL es porque el dato no existe en l */
                aux = l->sig;
                l->sig = aux->sig;
                free (aux);
                printf ("Dato %d eliminado\n", d);
            }
            else { /* Si l->sig es NULL */
                printf ("El dato %d no se encuentra en la lista\n", d);
            }
        }
    }
    return ret; /* Va a ser el nuevo principio de la lista */
}
```