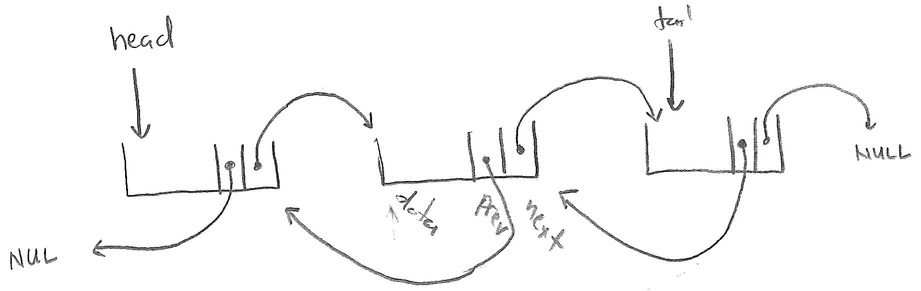
Doubly Linked List: List element contains pointers to next and previous element in list.

+provides a more flexible/intuitive interface for insertion and deletion
+provides ability to traverse the list forward and backward

General Doubly-linked List Structure:



Doubly-linked List Element:

```
typedef struct DListElmt_ {
        ListData  *data;
        struct DListElmt_ * next;
        struct DListElmt_ * prev;
} DListElmt;
```

* ListData should define a struct specifying what elements to store in the list

** Or, you can typedef ListData to an existing data type

Doubly-linked List:

```
typedef struct DList_ {
      DListElmt * head;
      DListElmt * tail;
      int size;
} DList;
```

Initialization:

```
void   dlist_init (DList * list) {
    list->head = NULL;
    list->tail = NULL;
    list->size = 0;
}
```

Destruction:

```
void   dlist_destroy (DList * list) {
    while (list->size > 0) {
        dlist_remove (list, list->head);
    }
}
```

Movement:

```
DListElmt * dlist_head (DList * list) {
        return list->head;
}

DListElmt* dlist_tail (DList * list) {
        return list->tail;
}

DListElmt* dlist_next (DListElmt * element) {
        return element->next;
}
DListElmt* dlist_prev (DListElmt * element) {
        return element->prev;
}
```

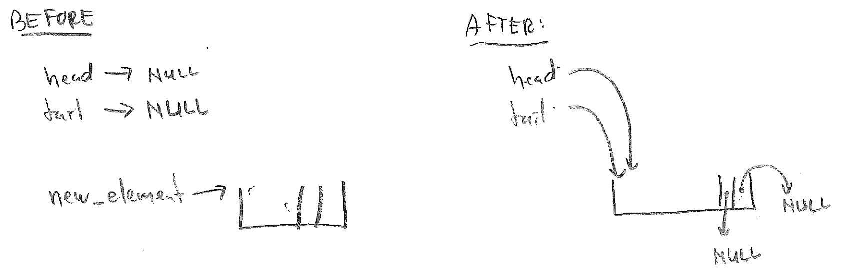
Size:

```
int  dlist_size (DList * list) {
     return list->size;
}
```

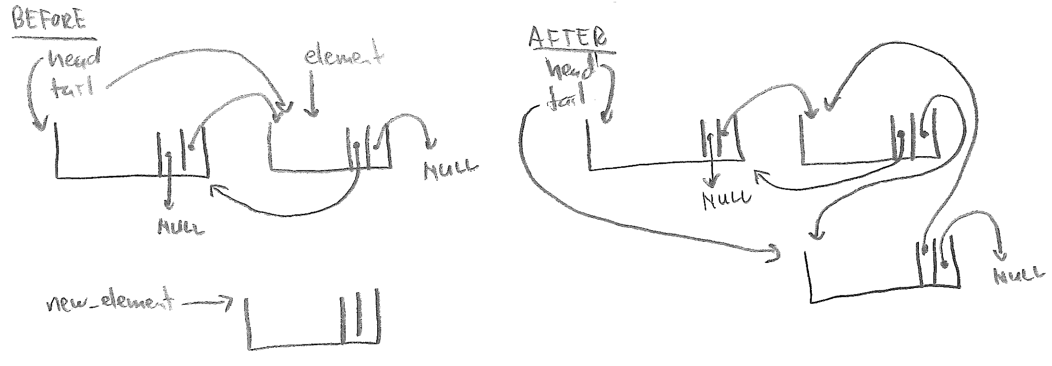
Insertion:
(pseudocode)

```
int dlist_insert_next (DList * list, DListElmt *element, ListData *data)

if element is null and list not empty, return error

malloc new element
if malloc failed, return error

assign new element data

if list is empty
    set list head to new element           }
    set list tail to new element           } case ①
    set element's next pointer to null     }
    set element's prev pointer to null     }
else
    set new element's prev pointer to element
    set new element's next pointer to element's next pointer

    if element is tail, set list tail to new-element          } case ②
    else set prev pointer of next element to new element      } case ③

    set element's next pointer to new element
increase size of list
return success
```

① empty list

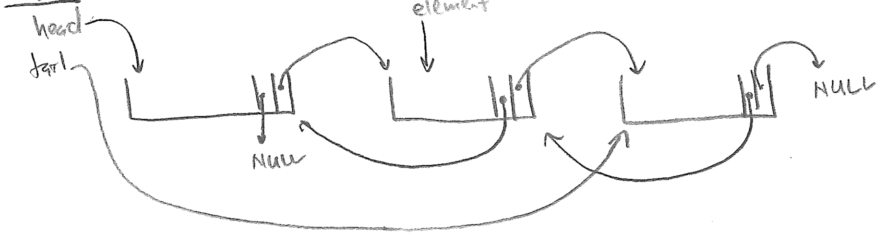


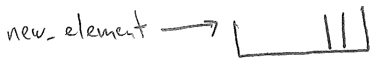
② insert after current tail of list



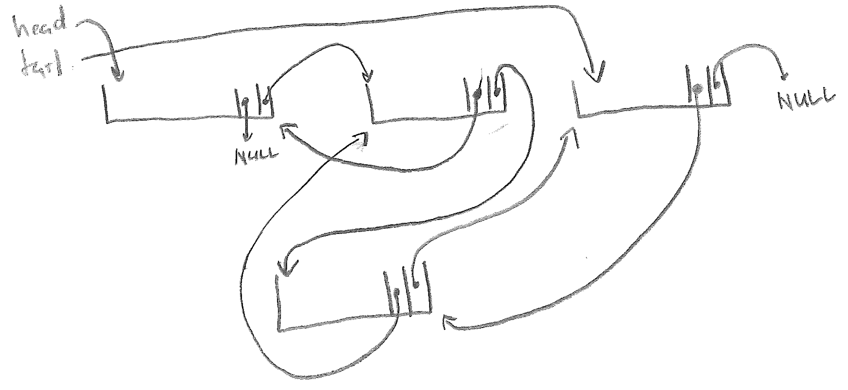
(3) insert in middle of list

BEFORE



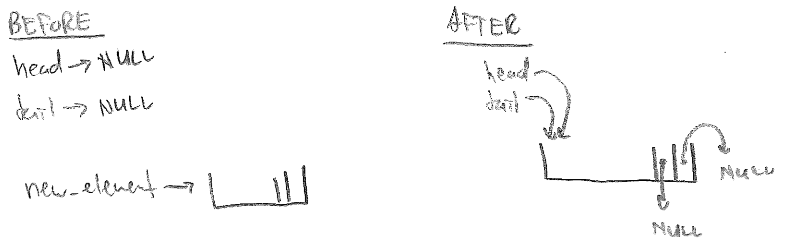


AFTER

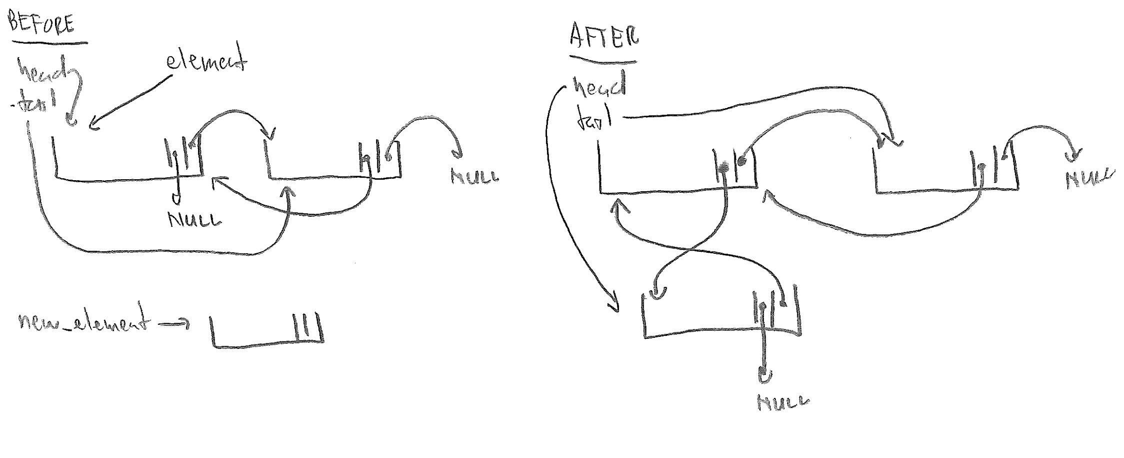


```
int dlist_insert_prev (DList *list, DListElmt *element, ListData *data)
    if element is null and list not empty, return error

     malloc  new list element
     if malloc failed, return error

     ~~assign~~ assign  new element data

     if list is empty
         set head to new element                          }
         set tail to new element                          } case ①
         set element's next and prev pointers to null     }
     else
         set new element's next pointer to element
         set new_element's prev pointer to element's prev pointer

         if element is head, set head to new_element           } case ②
         else  set next pointer of prev element to new_element } case ③

         set element's prev pointer to new-element

     increase size
     return success
```

① empty list

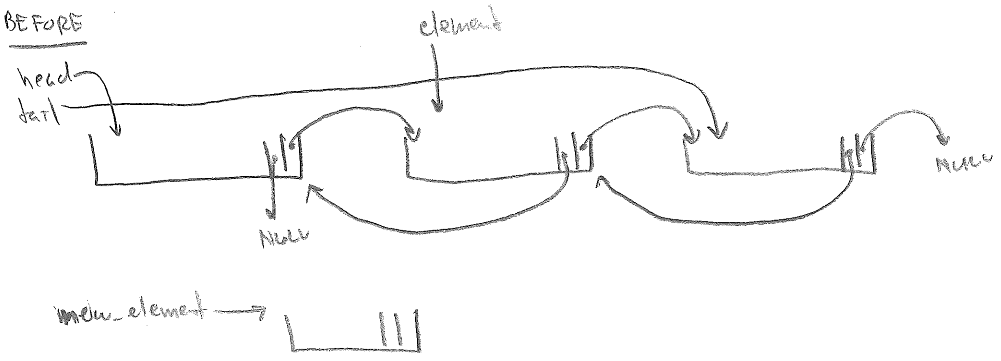


② insert before current head

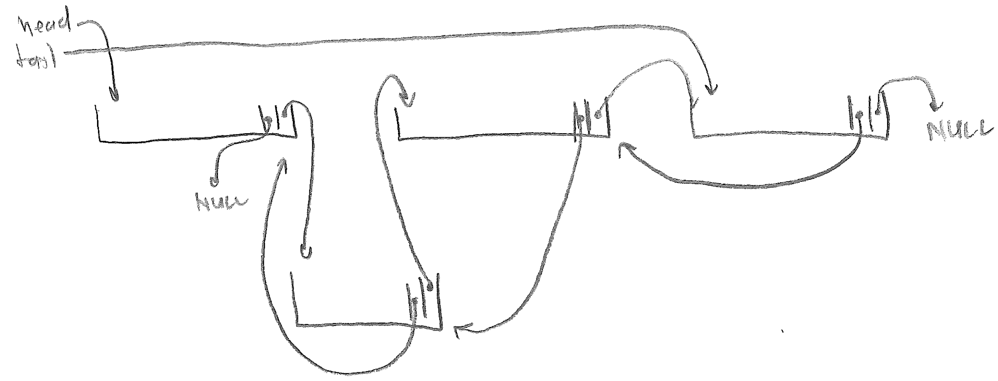


③ insert in middle of list

BEFORE



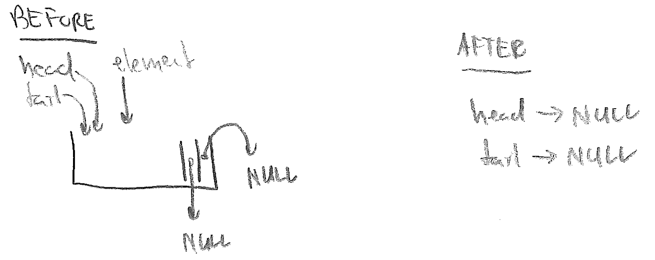
AFTER



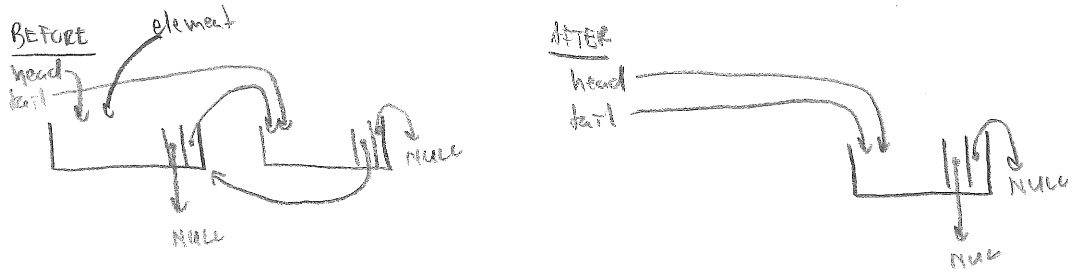
Removal: int list_remove (DList *list, DListElmt *element); removes element from list
(pseudocode)

```
if element is null or list is empty, return error
if element is head:
    set head to next element
    if list is now empty                                  } case ①
        set tail to null
    else
        set next element's prev pointer to null           } case ②
else
    set previous element's next pointer to element's next pointer
    if element is tail                                    } case ③
        set tail to previous element
    else
        set next element's prev pointer to element's prev pointer } case ④.
free element's memory (and data).
decrease size
```

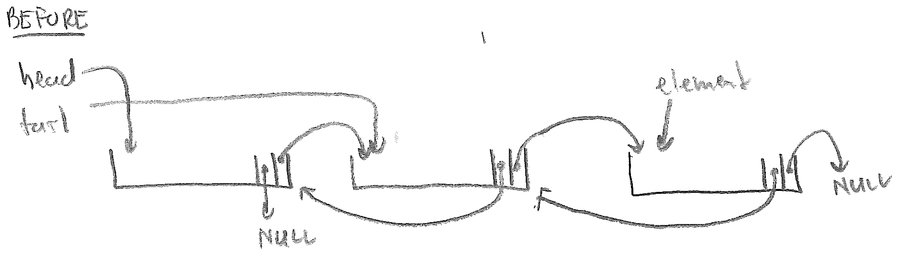
① remove last entry

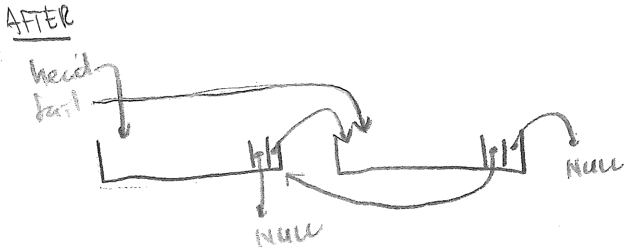


② remove head from list with more than one element

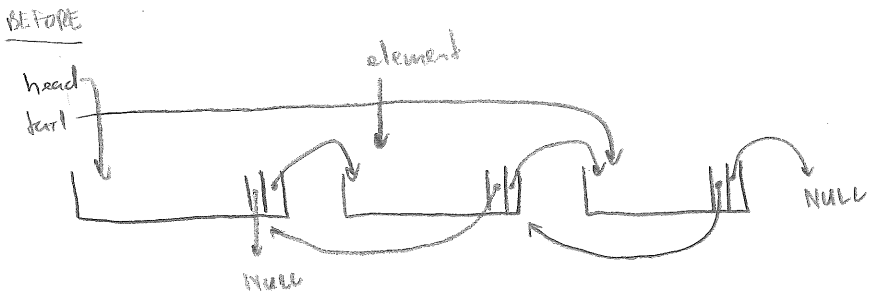


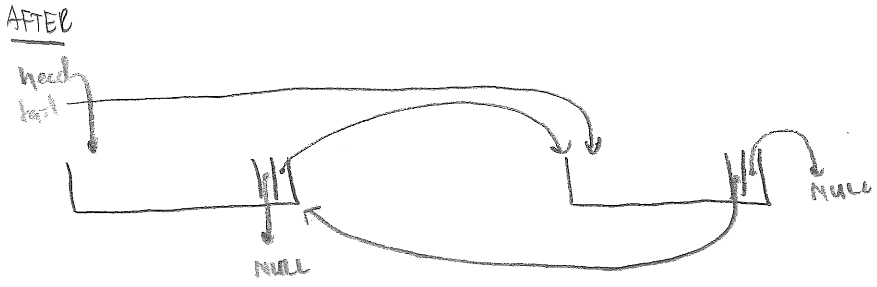
③ remove tail from lst with more than one element





④ remove from middle of lst


BEFORE
head
tail
element
NULL
NULL


AFTER
head
tail
NULL
NULL