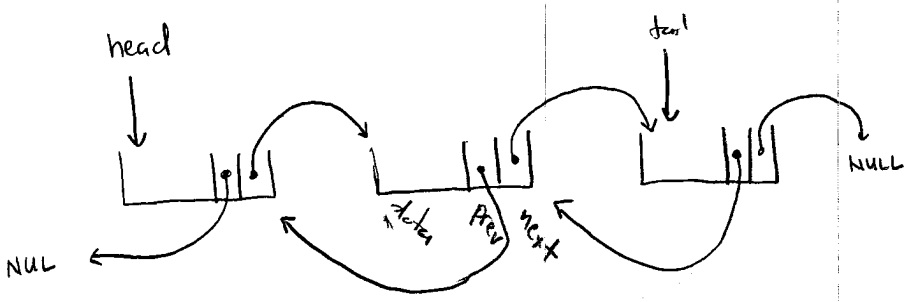
Doubly Linked List: List element contains pointers to next and previous element in list.

+ 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_ {
    int val;
    struct DListElmt_ * next;
    struct DListElmt_ * prev;
} DListElmt;
```

* we are creating a list of integers

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;
}
```

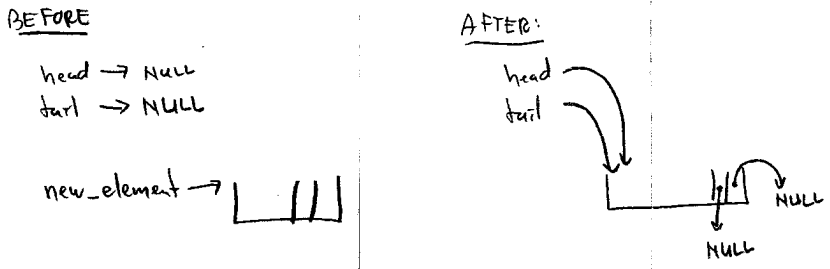
Size:

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

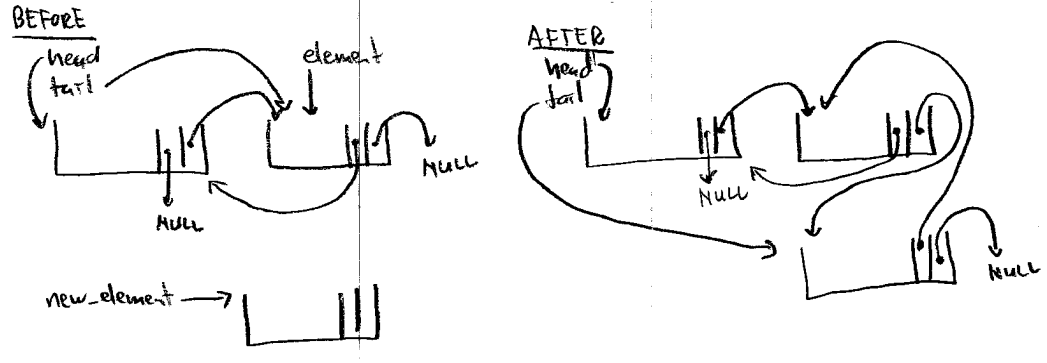
Insertion: (pseudocode)

```
int lst_insert_next (DLst * lst, DLstElmt *element, int val);

if element is null and lst not empty, return error

malloc new element
if malloc failed, return error

initialize new element data

if lst is empty
    set lst head to new element
    set lst tail to new element
    set element's next pointer to null
    set element's prev pointer to null                    } case (1)
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 lst 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 lst
return success
```

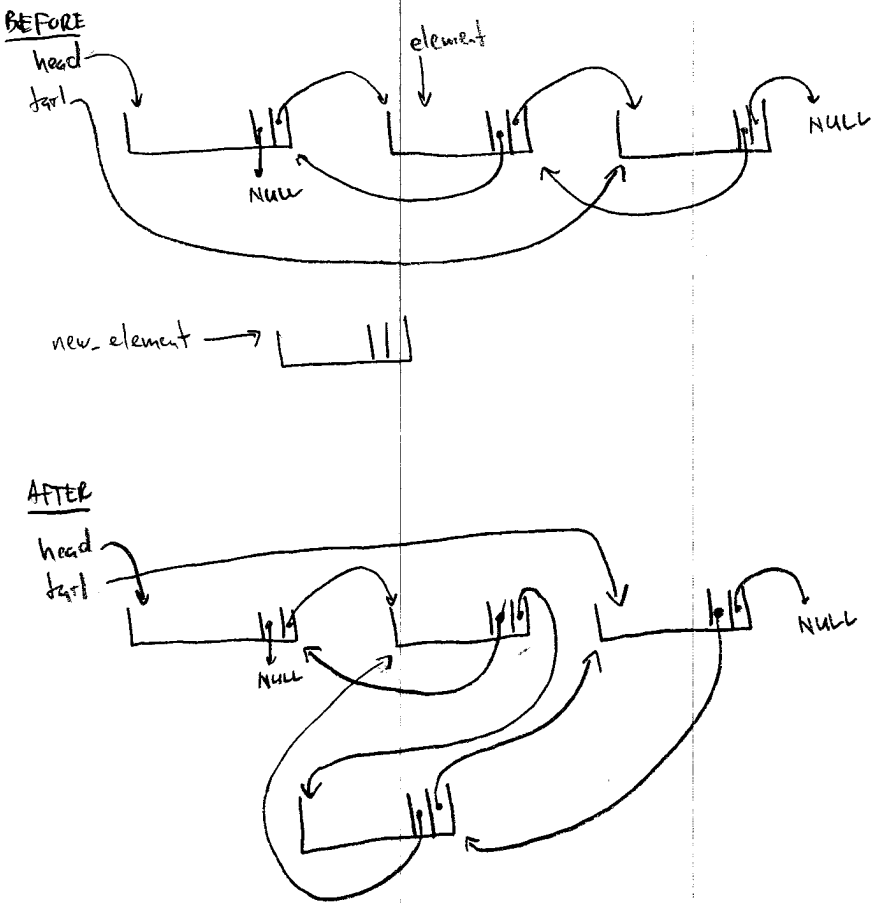
(1) empty list



(2) insert after current tail of list



③ insert in middle of list



int dlist_insert_prev (DList *list, DListElmt *element, int val);

if element is null and list not empty, return error

malloc new list element
if malloc failed, return error

initialize new element data

if list is empty

set head to new element
set tail to new element
set element's next and prev pointers to null } case ①

else

set new element's next pointer to element
set new_element's prev pointer to element's prev pointer

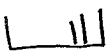
if element is head, set head to new_element } case ②
else set next pointer of prev element to new_element } case ③

set element's prev pointer to new element

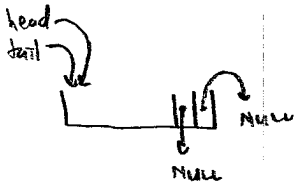
decrease size
return success

① empty list

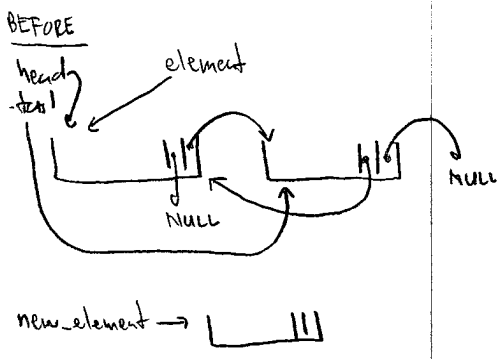
BEFORE
head → NULL
tail → NULL

new_element → 

AFTER

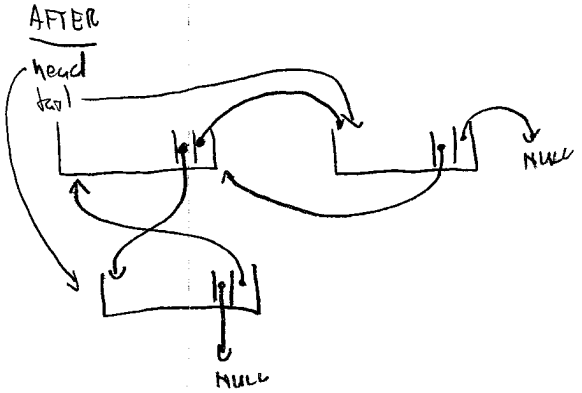


② insert before current head

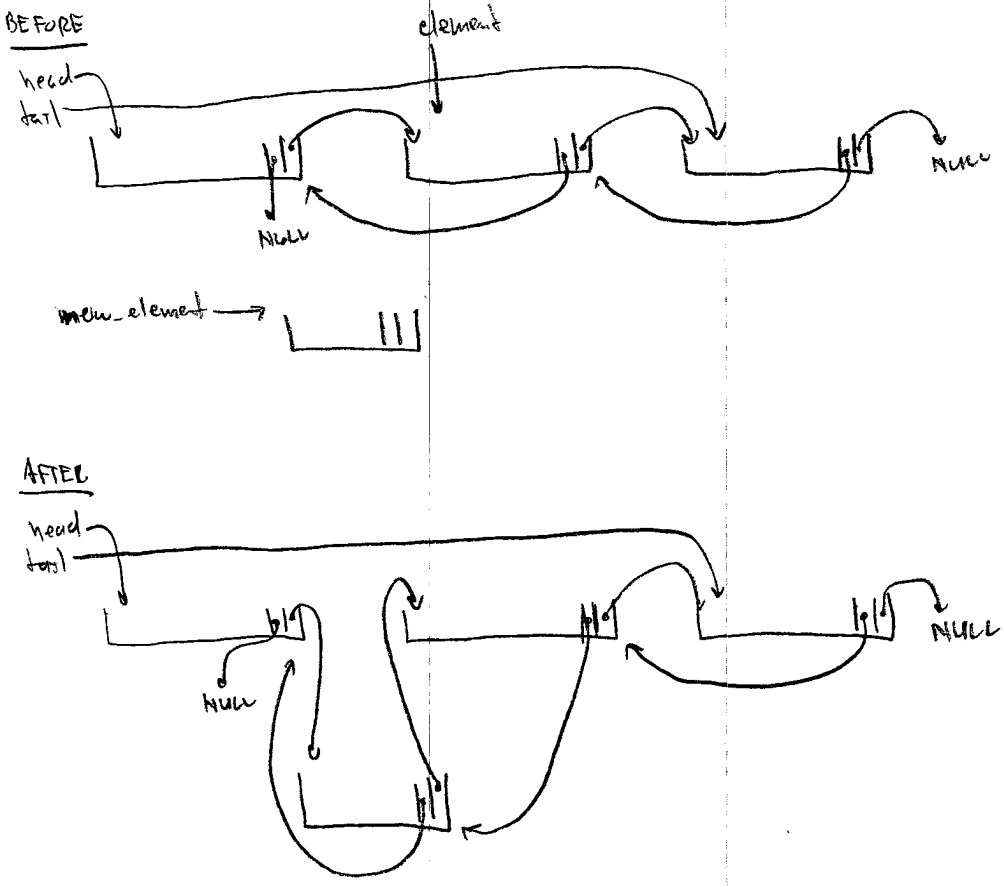


new_element →

AFTER



③ insert in middle of list



Removal: (pseudocode)

int lst_remove (DLst *lst, DLstElmt *element); removes element from lst

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 } case ②

set next element's prev pointer to null

else

set previous element's next pointer to element's next pointer

If element is tail } case ③

set tail to previous element

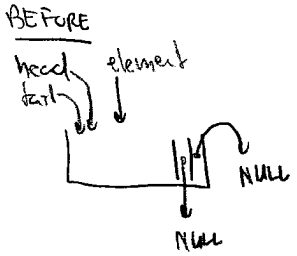
else

set next element's prev pointer to element's prev pointer } case ④

free element's memory

decrease list size

① remove last entry

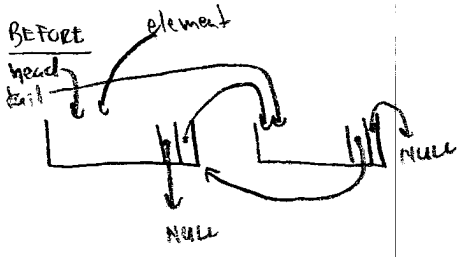


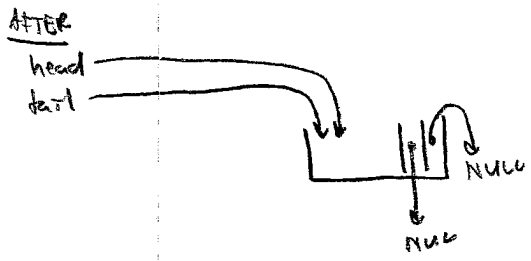
AFTER

head → NULL

tail → NULL

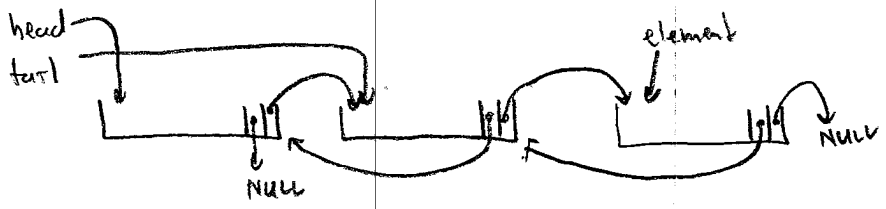
② remove head from list with more than one element



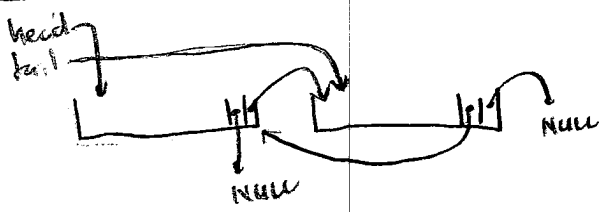


③ remove tail from list with more than one element

BEFORE

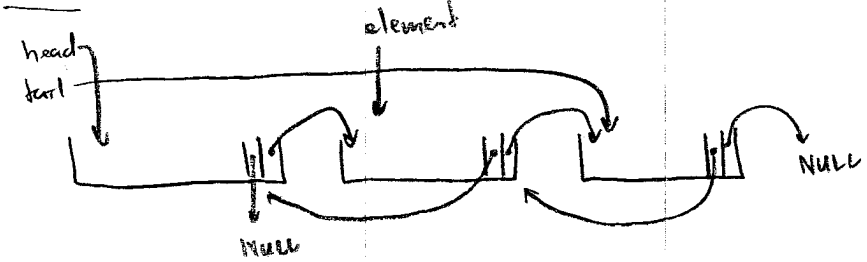


AFTER



④ remove from middle of list

BEFORE



AFTER