Lineare Liste:

⇒ Häufigste dynamische Datenstruktur

Vorteile gegenüber Arrays:
- wachsen & schrumpfen
- einfaches einfügen & löschen

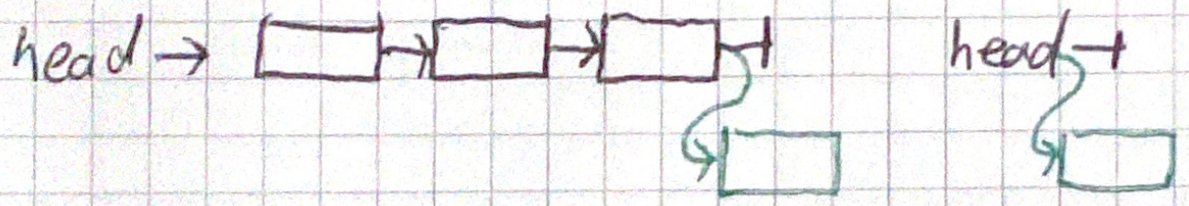
Nachteile gegenüber Arrays:
- kein indizierter Zugriff

```
Struct Node {
    Struct Node * next;
    int value;
};

Struct List {
    Struct Node * head;
};

int main () {
    Struct List * List = (struct List *) malloc ( sizeof(struct List));
    List -> head = NULL;
}
```

Einfügen am Listenanfang:

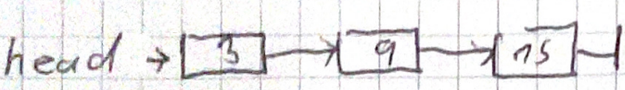

```
void insert ( struct List *list, int value) {
    struct Node * n = (...) malloc (sizeof (struct Node));
    n -> value = value;   n -> next;
    n -> next = list -> head;
    List -> head = n;
}
```

Einfügen am Ende:




```
void insert (struct List *list, int value) {
    struct Node * n = (...) malloc(sizeof(struct Node));
    n -> value = value; n -> next = NULL;
    if (list -> head == NULL) {
        list -> head = n;
        return;
    }
    struct Node * p = list -> head;
    while(p -> next != NULL) {
        p = p -> next;
    }
    p -> next = n;
}
```

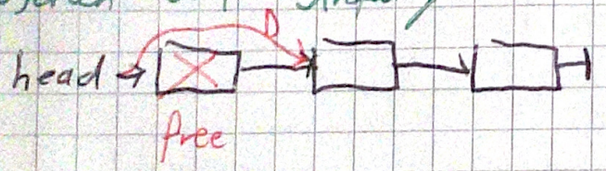
Suchen:




```
Struct node* search(struct List *list, int val){
    struct node *p = list→head;
    while(p!=NULL && p→value != val){
        p=p→next;
    }
    return p;
}
```

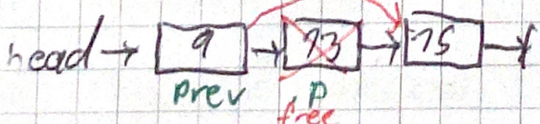
Tipp rekursion?

Löschen am Anfang:




```
void remove(Struct List *list){
    if(list→head != NULL{
        Struct list *p = List→head;
        list→head = list→head→next;
        free(p);
    }
}
```

Löschen eines Knotens mit Schlüssel val



val = 13

```
void remove(struct List *list, int val){
    struct Node *p = list→head;
    struct Node *prev = NULL;
    while(p!=NULL && p→val != val){
        prev = p;
        p=p→next;
    }
    if(p==NULL){
        return;
    }
```

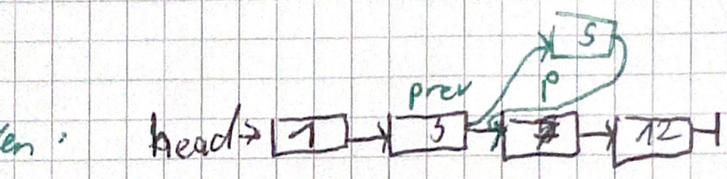
⇒ Liste Leer oder Wert nicht gefunden

nächste Seite

```
if(p==list->head){
    list->head = list->head->next;   ⇒ Löschen des 1ten Elements
    free(p);
}else{

    prev->next = p->next;
    free(p);
}
}
```

Sortierte Listen:

Einfügen (ohne Doppelte)




```
void insert(Struct List *list, int val){
    struct Node *p = list->head;
    Struct node *prev = NULL;
    while(p!=NULL && p->value < val){
        prev = p;
        p = p->next;
    }
    if(p==NULL || p->value != val){   // keine doppelten Einfügen
        struct node *n = (...)malloc(...);
        n->value = val;
        (n->next = NULL;)
        n->next = p;
        if(p==list->head){
            list->head = n;
        }else{
            prev->next = n;
        }
    }
}
```

TODO: Verkettete Liste

= welche Operationen auf Listen sind gut rekursiv zu lösen

- append, prepend
- insert(index)
- print()
- reverse
- count()
- insertSorted
- delete
- swap(index1, index2)
- sort

Löschen in der sortierten Liste.

```
void remove (struct List *list, int val) {
    Struct Node * p = list → head;
    Struct Node * prev * = NULL;

    while (p!
```