Algorithmen über einfach verketteten Listen

Listenelement:

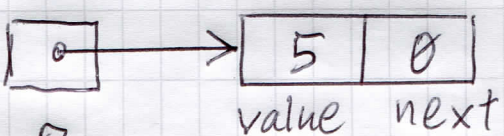

```
// in C:
struct el { int value; // Daten
            struct el *next; // Zeiger auf struct el
};
```

Anker, Anfangszeiger:

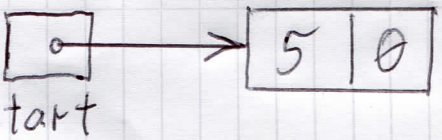

```
struct el *start = 0; // Initialisierung
```

Einfügen neues Listenelement am Anfang:

```
struct el *p = 0;   // neues Listenelement p
p = (struct el *)malloc(sizeof(struct el));
    // Speicherplatzzuweisung an p auf heap
p -> value = 5; // Wert 5
p -> next = 0; // kein Nachfolger
```




```
Insertfirst (↕start, ↓p){ //Funktion zum Einfügen
        p -> next := start
        start := p
}
```



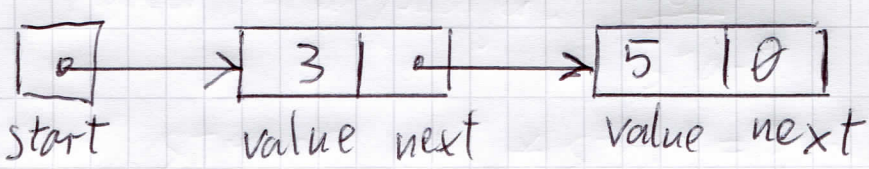
(2) // Listenelement p an den Anfang der Liste nach start:

```
Insertfirst(&start, p); //Aufruf Funktion
```

// weiteres Listenelement erzeugen:

```
p=(struct el *) malloc(sizeof(struct el));
p -> value = 3; // Wert 3
p -> next = 0; // kein Nachfolger
```

```
Insertfirst(&start,p); // Einfügen
```

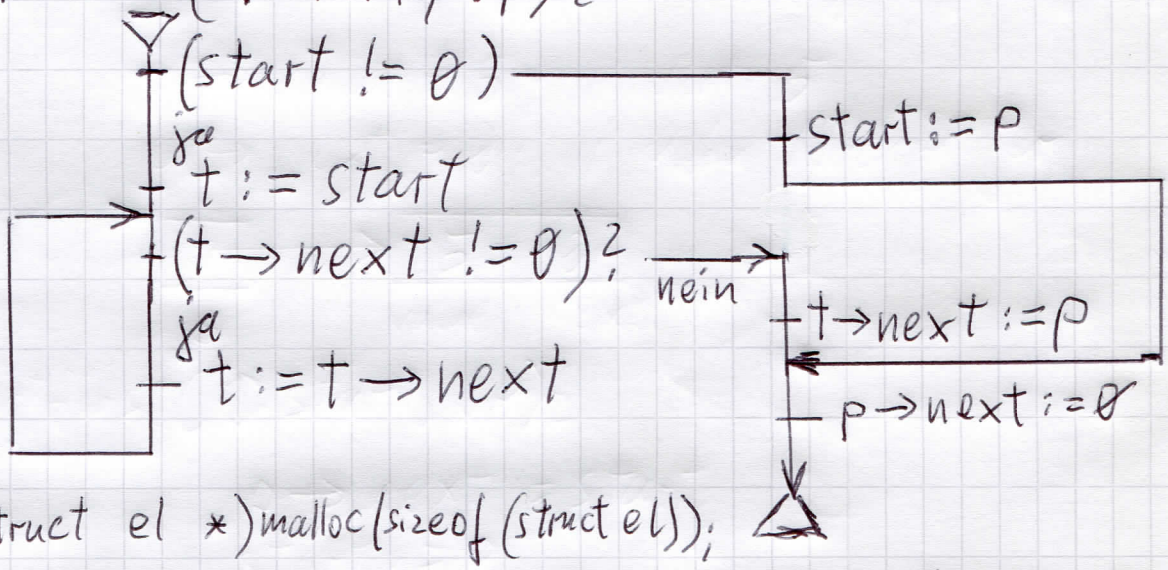


Erzeugen eines neuen Listenelementes und das Einfügen am Anfang kann beliebig oft erfolgen

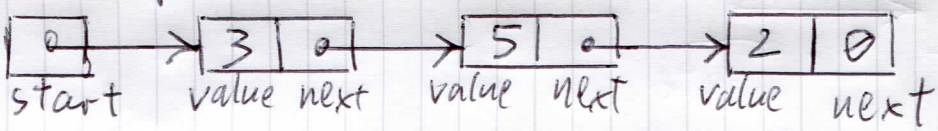
---

Einfügen neues Listenelement am Listenende:

```
Insertlast (↕start, ↓p){
```

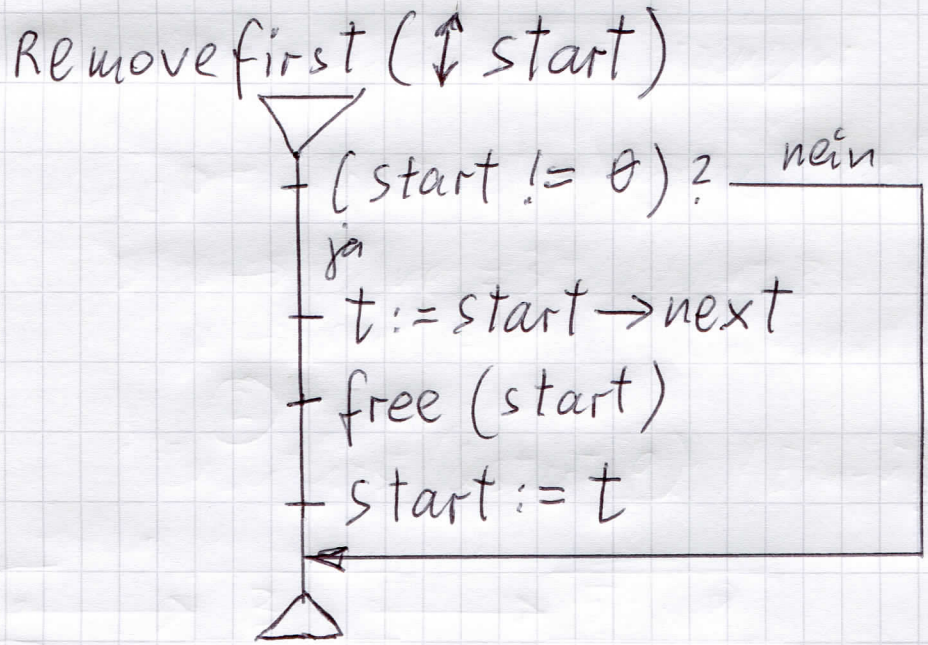



```
p=(struct el *)malloc(sizeof(struct el));
p -> value = 2; p -> next = 0; Insertlast(&start, p);
```

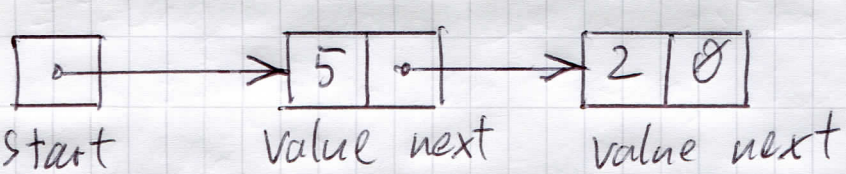


(3)

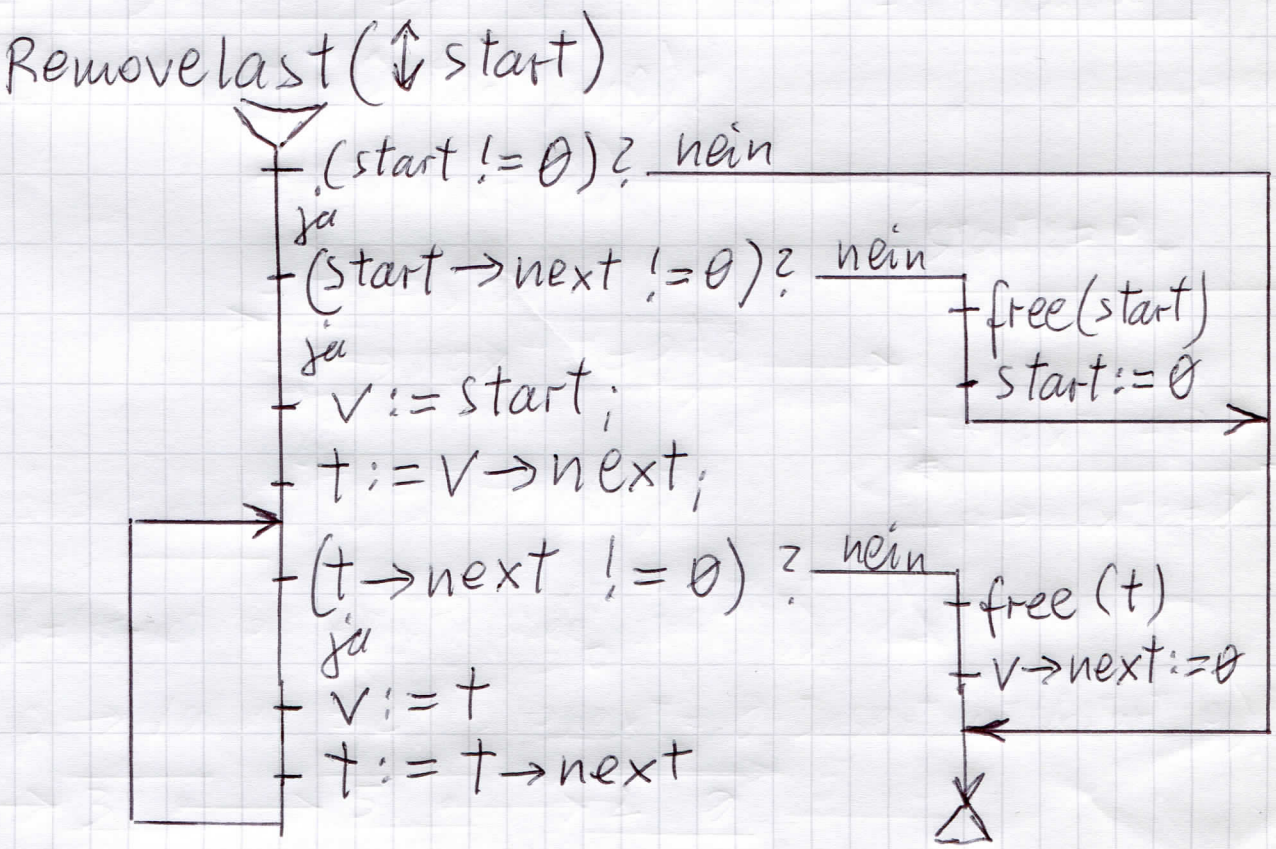
Löschen erstes Element nach start:



Remove first (& start);

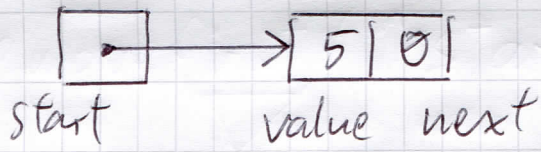


Löschen des letzten Elements



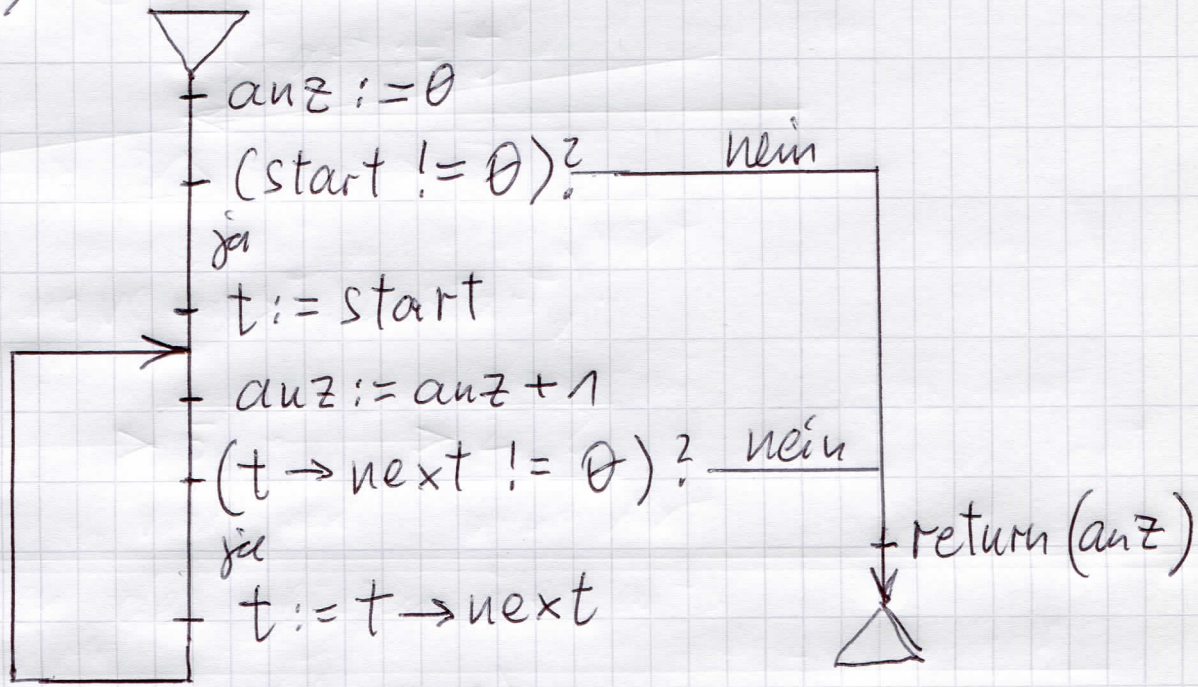
(4)

Removelast(&start)



Berechnung Anzahl Listenelemente:

(↑anz) Anzahl (↓start)

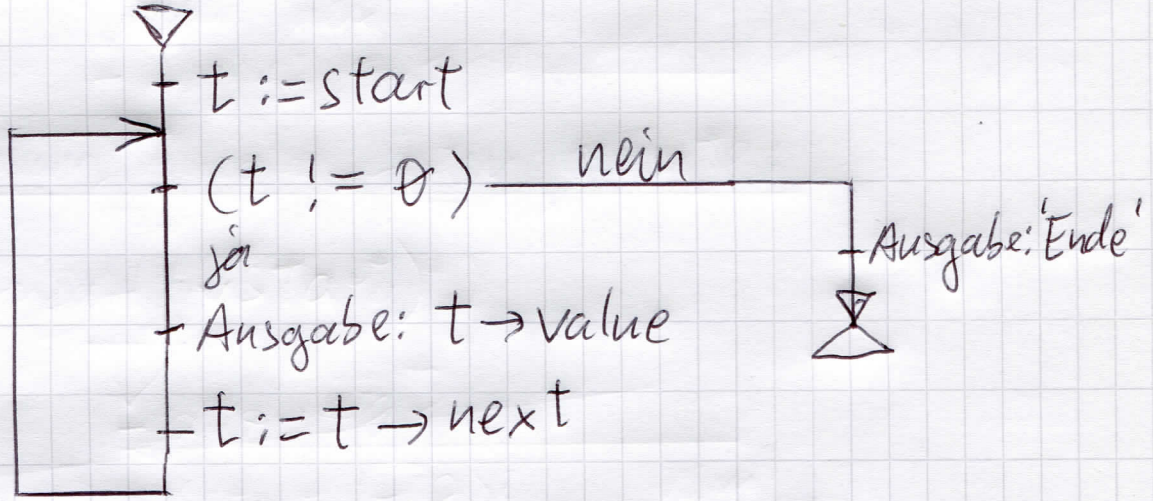


n = Anzahl(start); // 1

(5)

Anzeigen aller Listenelemente:

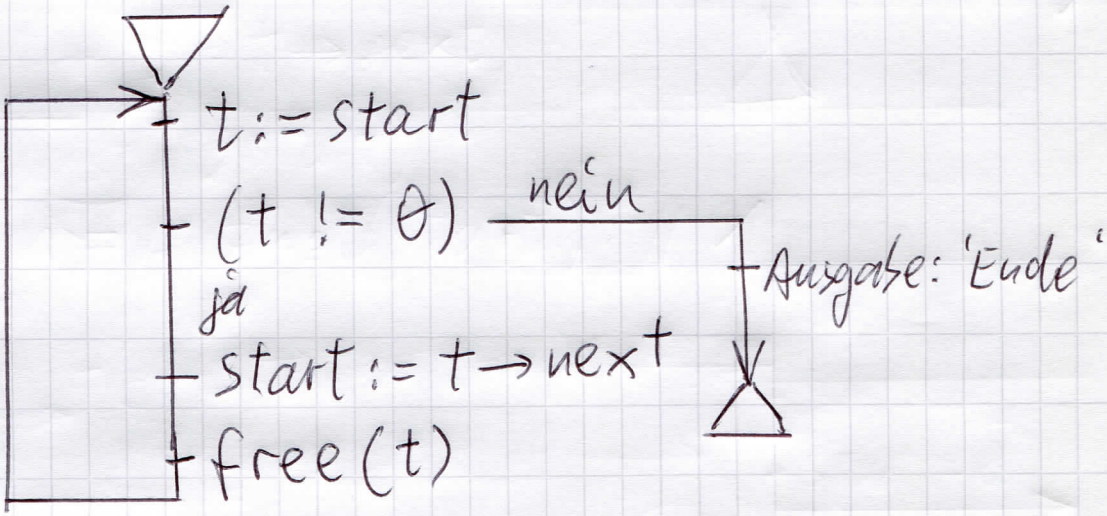
Show (↓start)



show(start);

Löschen aller Listenelemente:

Removeall (↕start)



Removeall(&start);