List Element x = new List Element();

Aufgabe 2: Verkettete Listen (3 Punkte)

Welche Wirkung haben die folgenden Codefragmente? Beschreiben Sie ihre Antwort in maximal zwei Sätzen und zeichnen Sie ein kleines Beispiel zur Verdeutlichung des Effektes.

(a) Angenommen, `x` ist ein Knoten in einer verketteten Liste und nicht der letzte.

```
x.next = x.next.next;
```

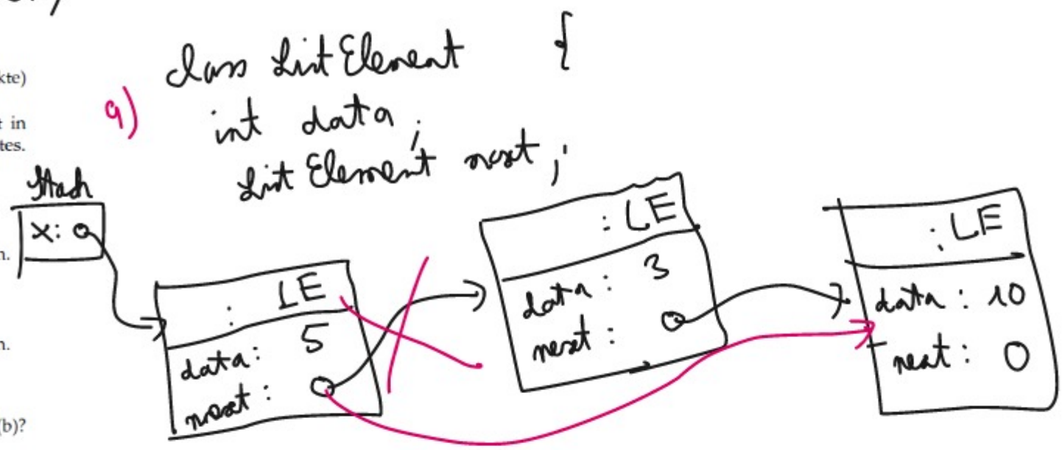
(b) Angenommen, `x` ist ein Knoten in einer verketteten Liste und `t` ist ein neuer Knoten.

```
t.next = x.next;
x.next = t;
```

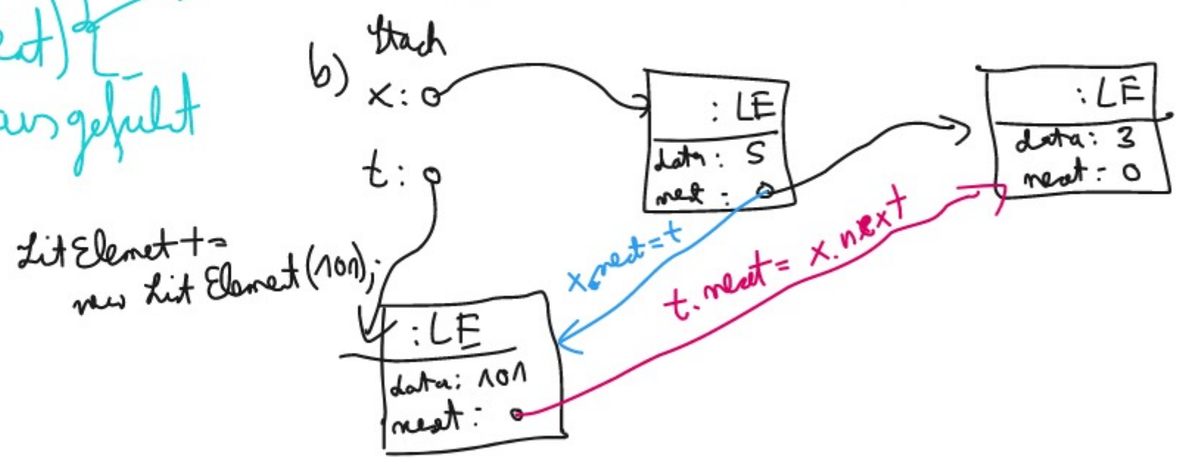
(c) Angenommen, `x` ist ein Knoten in einer verketteten Liste und `t` ist ein neuer Knoten.

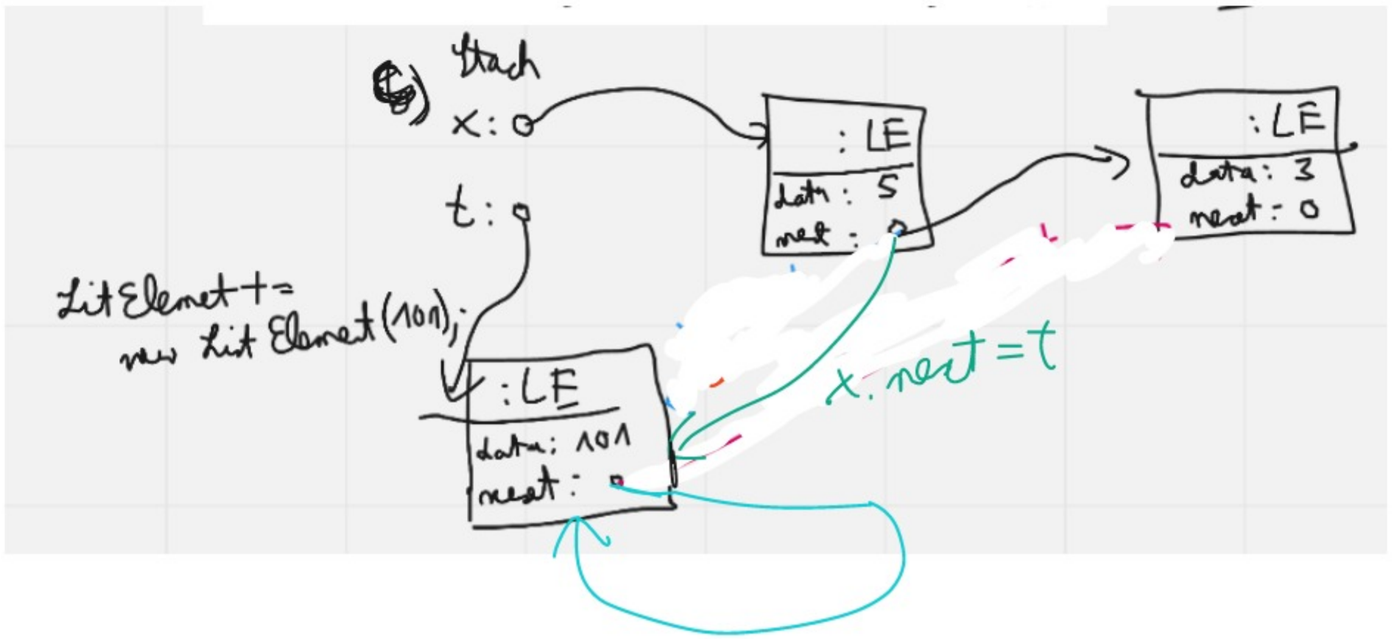
```
x.next = t;
t.next = x.next;
```

Warum bewirkt dieses Codefragment nicht das Gleiche wie das Codefragment aus (b)?



if(t == t.next){
→ wird ausgeführt
3





Aufgabe 3: Einfach Verkettete Listen (5+3 Punkte)

In dieser Aufgabe sollen Sie einfach verkettete Listen implementieren.

(a) Schreiben Sie eine Klasse `List<T extends Comparable<T>>` zur Repräsentation von einfach verketteten Listen. Ihre Klasse soll folgende Methoden besitzen:

- `boolean empty()`: Ist genau dann true, wenn die Liste leer ist. Laufzeit: $O(1)$.
- `int size()`: Gibt die Anzahl der Listenelemente zurück. Laufzeit: $O(1)$.
- `void clear()`: Löscht alle Listenelemente. Laufzeit: $O(1)$.
- `void append(T t)`: Fügt ein neues Element mit Wert `t` am Ende an. Laufzeit: $O(1)$.
- `void deleteLast()`: Löscht das Element am Ende der Liste. Laufzeit: $O(1)$.
- `void concatenate(List<T> l)`: Hängt `l` ans Ende der Liste an. Laufzeit: $O(1)$.
- `String toString()`: Ausgabe der Liste in der Form `[22,5,3]`. Laufzeit: $O(\texttt{size}())$.
- `void reverse()`: Kehrt die Reihenfolge der Elemente um. Laufzeit: $O(\texttt{size}())$.
- `void delete(int k)`: Löscht das Element an Position k. Der Listenkopf entspricht dabei der Position k = 0. Laufzeit: $O(k)$.
- `int deleteAllSmallerElements(T t)`: Löscht alle Elemente mit Wert $x < t$ aus der Liste und gibt die Anzahl der gelöschten Elemente zurück. Laufzeit: $O(\texttt{size}())$.

Beachten Sie bei Ihrer Implementierung die angegebenen Laufzeiten.

```
class ListElement<T extends Comparable<T> >{
   public T data;
   public ListElement<T> next;
}
```

List<Integer> l = new List<Integer>();

class List<T> { public ListElement last; Stack l: → :List head: → :ListElement data: 100 next: → :ListElement data: 300 next: 0

public ListElement head;
public int size = 0;
public boolean empty(){ return head == null; } boolean b (head==null) return b;

public int size(){ int s = 0; ListElement x = head; while(x != null){ x = x.next; ++s; } return s; } ∈ O(n)

public int size(){ return size; } ∈ O(1)

public void clear(){ head = null; size = 0; last = null; } ∈ O(1)

void append(T t){ ∈ O(n)
  size++;
  ListElement<T> node = new ListElement<T>();
  node.data = t;
  ListElement<T> x = x.head;
  while(x.next != null){ x = x.next; }
  x.next = node;
}

Stack: node: → this: → :ListEle data: t next: 0

this.head.next.next = node

public void append(T t){ ∈ O(1)
  size++;
  ListElement<T> node = new ListElement<T>();
  node.data = t;
  last.next = node;
  last = node;
}

public void deleteLast(){ if(size > 1){
  last.previous.next = null
  last = last.previous;
  --size; return; }
}

Stack: this: → :LE next: → :LE next: → :LE next: → :LE data: 100 next: 0 previous: →

class ListElement<T> { ListElement<T> next; ListElement<T> previous;