FHV

Algorithmen & Datenstrukturen

Listen

Wolfgang Auer

Liste

- Eine Liste ist eine *Datenstruktur* zur Verwaltung einer *Sequenz* von Elementen,
  - die über eine natürliche Ordnung verfügt
  - für die die folgenden Operationen definiert sind:
    - Erzeugen einer Liste
    - Zugriff auf ein Element
    - Einfügen eines Elements
    - Löschen eines Elements
    - Bestimmen der Länge einer Liste
    - Test auf „Leere Liste“

- Eine Liste L ist eine Menge von geordneten Elementen:

  $L = (x_0, x_1, x_2, ..., x_{n-1})$

  Länge $|L| = |(x_0, x_1, x_2, ..., x_{n-1})| = n$

  Eine leere Liste hat die Länge 0

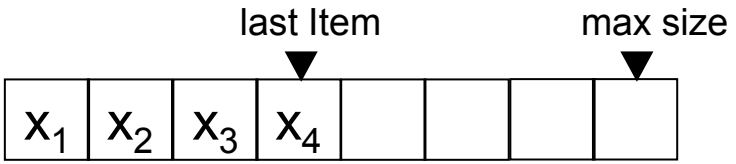
Repräsentation einer Liste

FHV

- Abhängig von der Verwendung, der gewünschten Effizienz und dem Speicherplatzbedarf können unterschiedliche Repräsentationen für eine Liste gewählt werden
  - Feld (Array)
  - Einfach verkettete Liste
  - Doppelt verkettete Liste
  - Zirkuläre Liste
  - ...

Feld (Array)

- Eine Feld ist eine *statische* Datenstruktur d.h. nach der Erzeugung kann die Länge nicht mehr verändert werden.



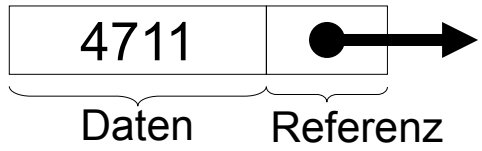
- Die Speicherung der Element erfolgt in einem zusammenhängenden Speicherbereich (*contiguous memory*)

- Vor- und Nachteile
  - ▲ Es wird kein zusätzlicher Speicher für die Verkettung der Elemente benötigt
  - ▲ Zugriff auf Elemente mit bekanntem Index erfolgt in O(1)
  - ▼ Einfügen und Löschen (Ausnahme am Listenende) aufwendig
  - ▼ Länge der Liste ist nach Initialisierung unveränderbar
  - ▼ Listendefinition verlangt keine Operation, die prüft, ob noch Listenplätze frei sind => Überlauf ist möglich

Verkettete Liste (1)

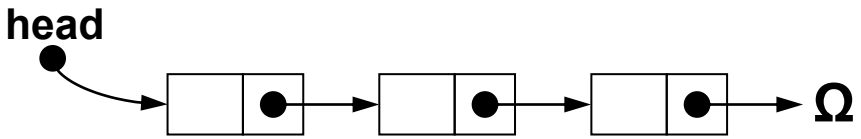
- Eine *verkettete Liste* (linked list) ist eine *dynamische* Datenstruktur zur Verwaltung einer Sequenz von Elementen mit folgenden Eigenschaften:
  - die Größe kann sich zur Laufzeit dynamisch ändern
  - einfaches Einfügen und Löschen von Elementen
  - direkter Zugriff auf einzelne Elemente ist **nicht** möglich

- Jedes Listenelement (Knoten) besteht aus zwei Komponenten
  - *Daten*: Eigentliche Information, die verwalten werden soll
  - *Referenz*: Verweis auf das nächste Element



Verkettete Liste (2)

- Eine *verkettete Liste* (linked list) entsteht durch das Aneinanderfügung von Listenelementen (Knoten)



- Der Anfang wird mit Hilfe einer Referenz *head* markiert
- Das letzte Element verweist auf `null`

Verkettete Liste (3)

Knoten einer einfach-verketten Liste

Verwaltung von Ganzzahlenwerten

```
class Node {
      int value; /* Daten */
      Node next;
}
```

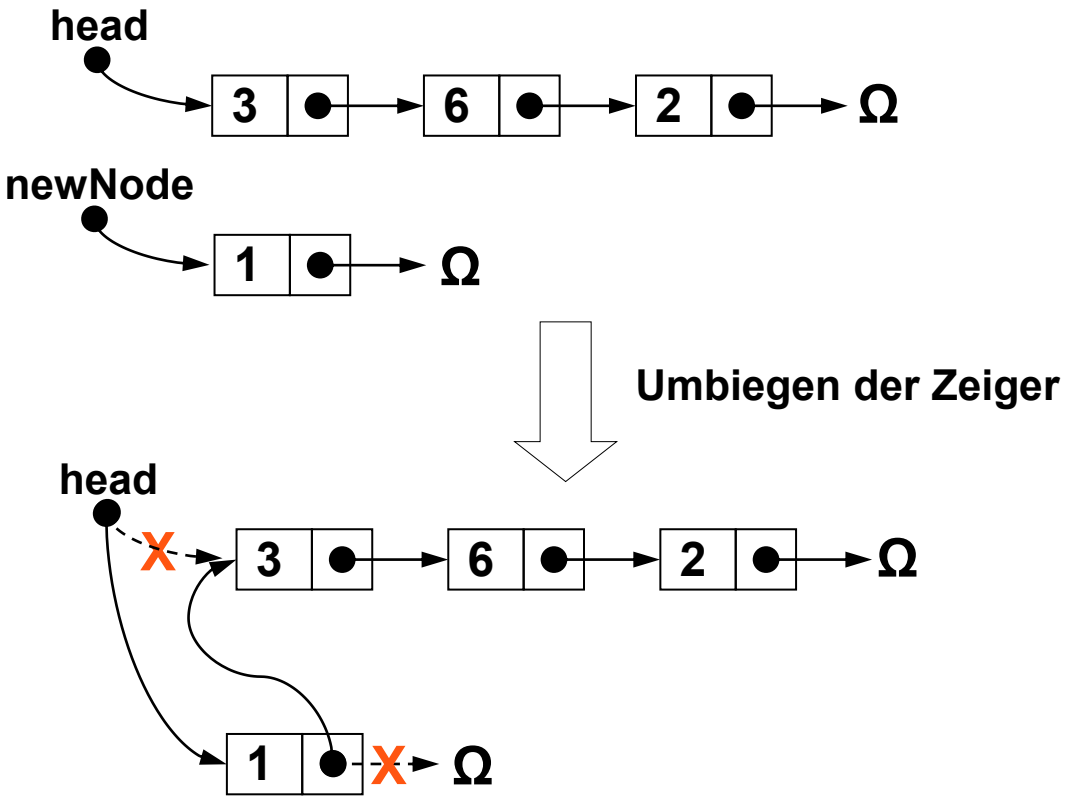
Verweis auf den nächsten Knoten, der vom gleichen Typ ist ⇒ *rekursive Datenstruktur*

Verwaltung von Punkten

```
class Node {
      Point value; /* Daten */
      Node next;
}
```

Verkettete Liste (4)
Einfügen eines Knotens am Kopf

- Allgemeiner Fall (Nicht leere Liste)



Verkettete Liste (5)
Einfügen eines Knotens am Kopf

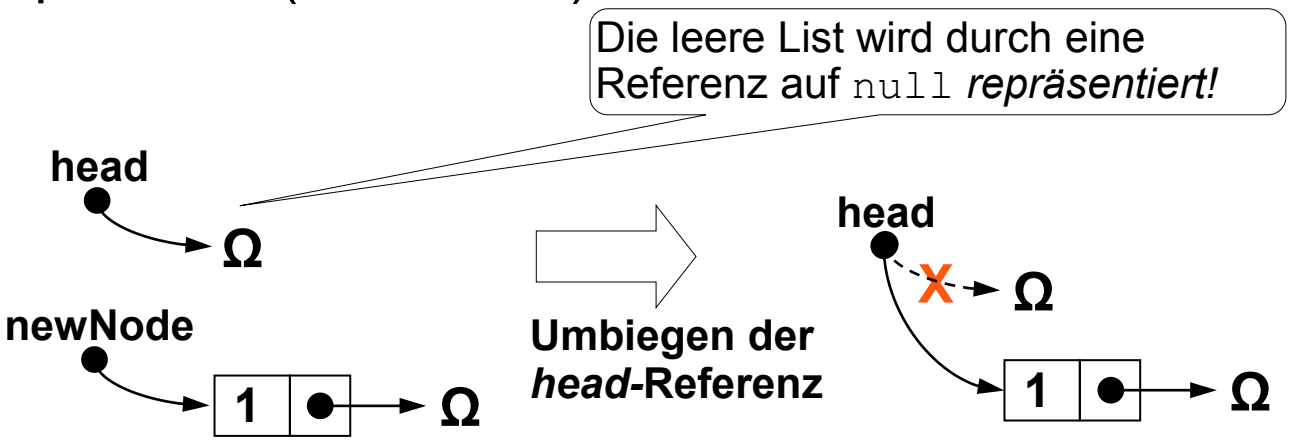
- Allgemeiner Fall (Nicht leere Liste)

```
/* Create new node */
Node newNode = new Node();
newNode.value = 1;
newNode.next = null;

/* Insert new node */
newNode.next = head;
head = newNode;
```

Verkettete Liste (6)
Einfügen eines Knotens am Kopf

- Spezialfall (leere Liste)

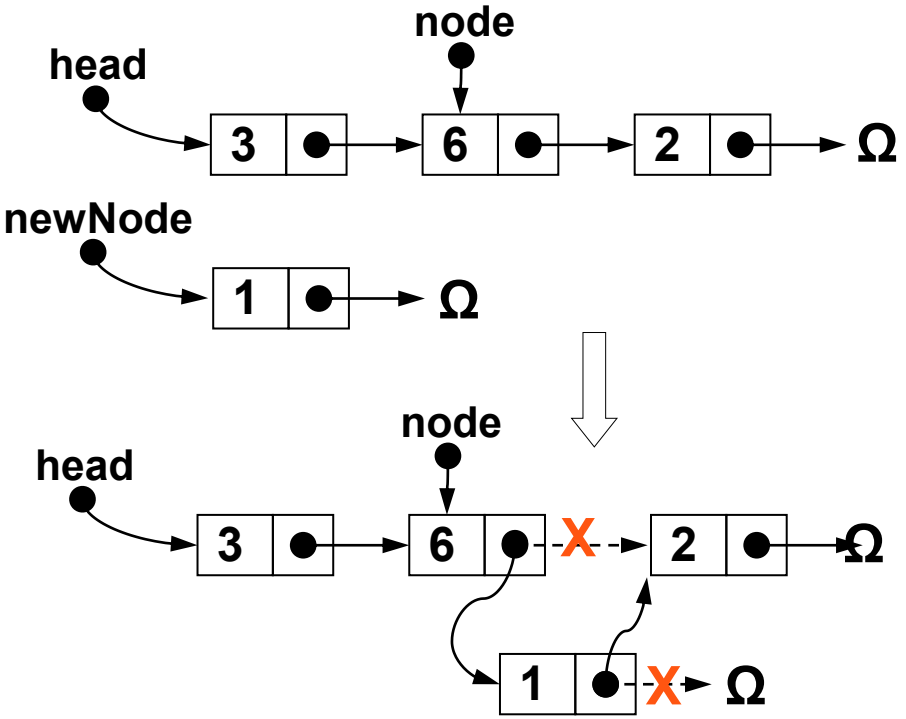



```
/* Create new node */
Node newNode = new Node();
newNode.value = 1;
newNode.next = null;

/* Insert new node */
head = newNode;
```

Verkettete Liste (8)
Einfügen eines Knotens

- Einfügen eines Knotens **hinter** einem Knoten `node`

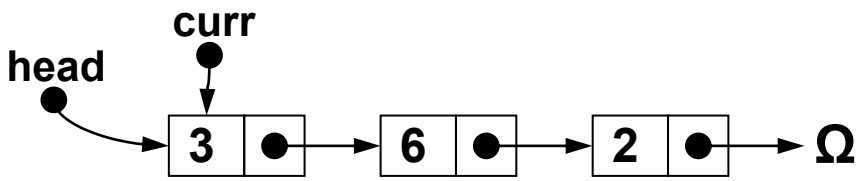



```
/* Insert after a node*/
newNode.next = node.next;
node.next = newNode;
```

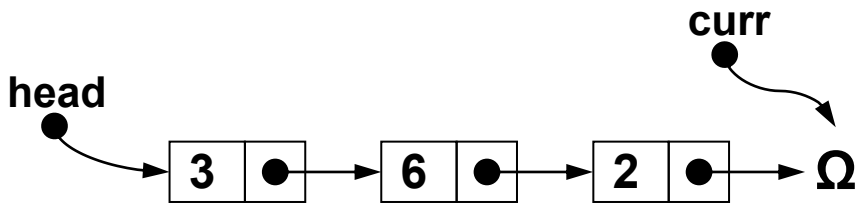
Verkettete Liste (9)
Durchlaufen einer Liste

```
Node curr = head;
while (curr != null) {
   sysout("Value = %d\n", curr.value);
   curr = curr.next;
}
```

Situation vor der Schleife

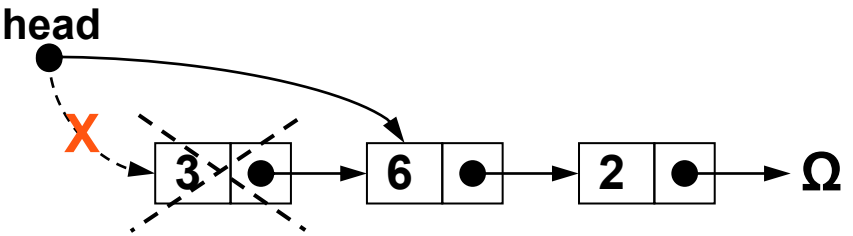


Situation nach der Iteration

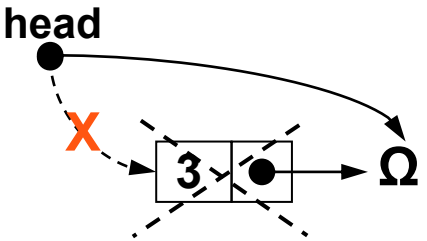


Verkettete Liste (10)
Löschen des ersten Elements

- Allgemeiner Fall (nicht leere Liste)



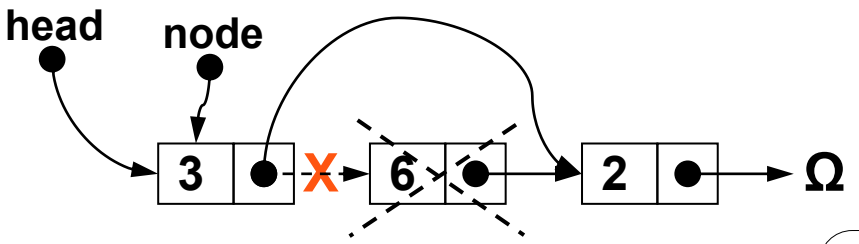
- Löschen aus einer Liste mit einem Element




```
temp = head;
head = head.next;
```

Verkettete Liste (11)
Löschen eines Knotens

- Löschen des Knotens **hinter** Knoten `node`



Achtung beim Löschen hinter dem letzten Element! => Sonderfall!

```
temp = node.next;
node.next = temp.next;
```

Verkettete Liste (12)
Bewertung

- Vor- und Nachteile
  - ▲ Länge kann sich während der Laufzeit dynamisch ändern
  - ▲ Einfügen und Löschen nach einem Element effizient
  - ▲ Keine Speicherverschwendung durch Vorreservierung von Speicherplatz

  - ▼ Zugriff auf Elemente ist sequentiell
  - ▼ Vorgänger eines Elements nur umständlich zu ermitteln
  - ▼ Extra Speicherplatzbedarf zur Verkettung

Marke (sentinel)

FHV

- *Sentinel* (Marke bzw. Wächter): **Dummy-Knoten**, der den Umgang mit Listen erleichtert d.h: Grenzfälle werden entschärft z.B.:
  - Kennzeichnung des Listenendes
  - Einführen eines speziellen Anfangsknotens,
    - um Informationen über die Liste zu verwalten
    - um das Entfernen des ersten Elements für den Fall, dass mehrere externe Zeiger auf den Listenanfang verweisen, zu vereinfachen

Das asymptotische Laufzeitverhalten wird durch die Verwendung von *Sentinels* nicht beeinflusst. Die effektive Laufzeit wird verbessert!