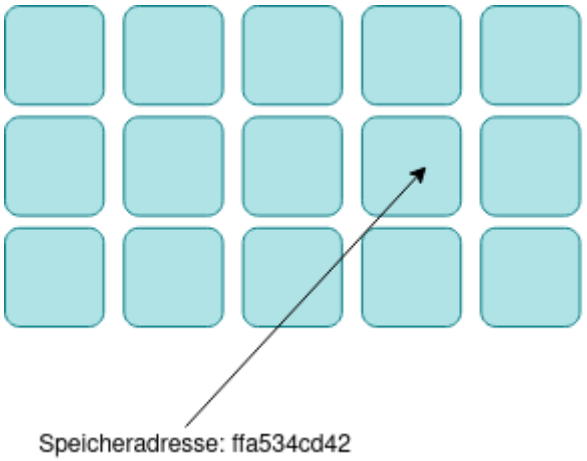
# Verkettete Listen

## Der Arbeitsspeicher

Man kann sich den Arbeitsspeicher eines Computers in etwas so vorstellen, wie eine Ansammlung von Aktenschränken mit Schubfächern:

Um darin etwas wiederzufinden, muss man die "Adresse" des Objekts kennen, z.B. Schrank 32, Fach 3, Register 9a, kurz S32F3R9a. Möchte man etwas ablegen, muss man sich entsprechenden Platz reservieren lassen, man erhält dann eine Adresse zurück, die man zur Ablage verwenden kann.

Ganz ähnlich verfährt der Computer: Wenn man eine Variable deklariert (oder ein anderes Objekt speichern möchte), wird Speicherplatz reserviert, in dem die Informationen gespeichert werden können. Dein Programm kennt die Adresse, an der die Informationen abgelegt sind. Bei modernen Programmiersprachen sieht man als Programmierer für gewöhnlich keine Speicheradressen mehr, die Verwaltung übernimmt der Compiler gemeinsam mit entsprechenden sprachlichen Konstrukten der Programmiersprache.




## Arrays und Listen

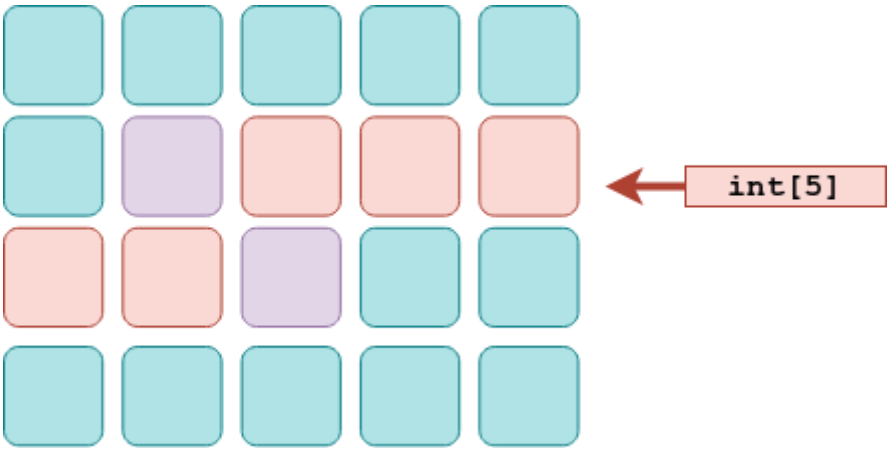
Wenn du mehrere gleichartige Objekte speichern möchtest, hast du im wesentlichen zwei Möglichkeiten: **Arrays** und **Listen**. Beide Datentypen haben Vor- und Nachteile.

# Arrays

Bei eineM Array legst du bereits zum Zeitpunkt der Deklaration fest, wieviele Elemente es enthalten soll:

```
int[] zahlenarray = new int[5];
```

Damit verbunden, ist die Anforderung eines bestimmten Speicherbereichs.

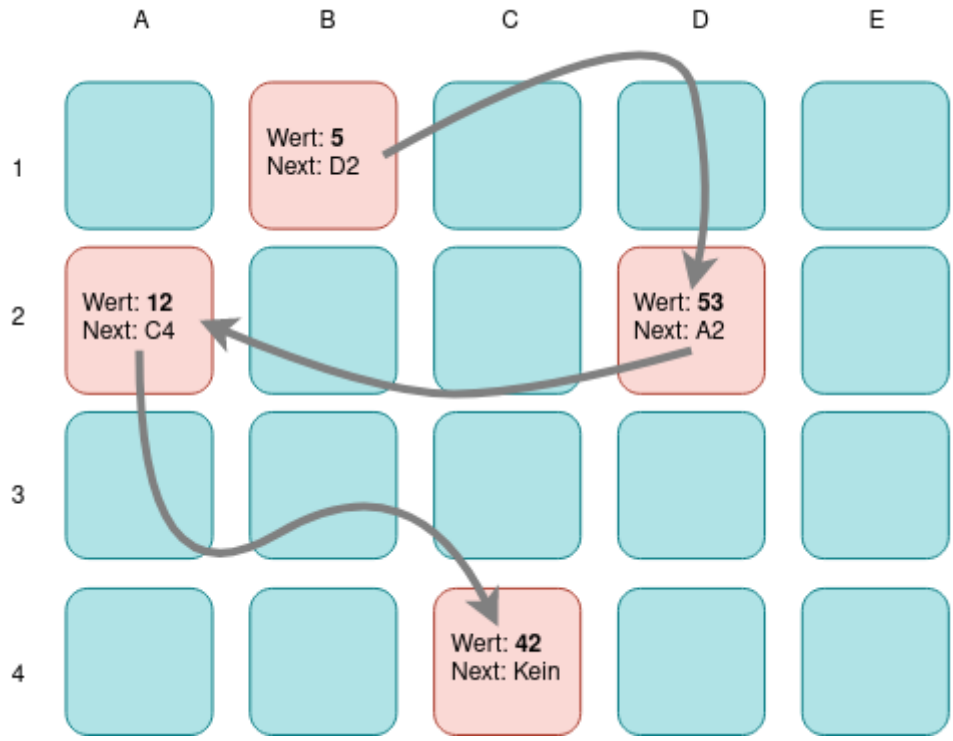



Eine spätere Erweiterung ist nicht vorgesehen - möglicherweise sind die Speicherbereiche vor und nach dem für das Array reservierten Speicher bereits anderweitig vergeben?

Denk an eine Reservierung im Kino: Wenn du 3 Plätze nebeneinander reserviert hast und kurz vor der Vorstellung nochmal zwei weitere Plätze direkt im Anschluss haben möchtest, klappt das meist nicht. Du musst in diesem Fall einen neuen Bereich suchen, der genügend Sitzplätze für alle deine Freunde hat und alle Plätze dorthin verschieben.

# Verkettete Listen

Bei der Speicherung als **verkettet Liste** können sich die Objekte überall im Hauptspeicher befinden, man kann jederzeit ein neues Element anfügen - wie geht denn das?




Eine verkettete Liste funktioniert ein wenig wie eine Schnitzeljagd. Jedes Element enthält die zu speichernde Information ("Wert") und die Adresse seines Nachfolgers. Wenn man nun das erste Element kennt, kann man Schritt für Schritt zu allen anderen Elementen gelangen.

---

**(A1)**

- Schreibe die Zahlenreihe aus dem Beispiel auf. Welche Adressen tauchen Dabei auf?
- Welche Nachteile gegenüber einem Array erkennst du?

# Operationen

Was wollen wir auf unseren Datenstrukturen (Array/Liste) gerne für Operationen ausführen?

- Lesen
- Einfügen (Am Ende/in der Mitte)
- Löschen

### Aufwandsabschätzung:

Der Aufwand für die Operationen ist folgender:

| | **Array** | **Liste** |
|---|---|---|
| Lesen | O(1) | O(n) |

| | **Array** | **Liste** |
|---|---|---|
| Einfügen | O(n) | O(1) |
| Löschen | O(n) | O(1) |

Zur Erinnerung: *O(1)* ist eine *konstante Laufzeit*, es hängt also nicht davon ab, wie lang die Liste ist oder wie viele Elemente das Array hat.

*O(n)* bedeutet *lineare Laufzeit*, d.h. die Operation dauert linear länger mit der Zahl der Elemente deiner Datenstruktur.

---

**(A2)**

Erkläre die unterschiedlichen Laufzeiten bei den verschiedenen Datenstrukturen. Überlege dir dazu, was man Schritt für Schritt tun muss, um z.B. ein Element in der Mitte einer Liste zu entfernen und was man machen muss, um das bei einem Array zu tun. Tipp: Das Array darf nach dem Vorgang keine "Löcher" haben.

# Darstellung