Einführung in die Softwareentwicklung

SOMMERSEMESTER 2020

Institut für Informatik

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

```
// This is C++
void main() {
   int var = 42;
   std::string s = "Hello World!\n";
   s += "The answer is still: ";
   s += string(var);
   std::cout << s;
}
```

Foliensatz #03

Datenstrukturen

Michael Wand · Institut für Informatik · michael.wand@uni-mainz.de

Zusammengesetzte Datentypen

Das Alte Beispiel…

…mit den Brüchen

structs & classes

```
struct Bruch {  // es gibt auch „class“; das ist fast das gleiche; dazu mehr später
    double zaehler;
    double nenner;
};
void main() {
    Bruch a; a.zaehler = 42; a.nenner = 23;
    Bruch b = a;        // „Wert-Semantik“! (Kopiert Werte, keine Refs!)
    a.zaehler = 1337; // Hier auch… (alle Zuweisungen kopieren Werte)
    a.zaehler -= 13;
    cout << a.zaehler << " / " << a.nenner << "\n"; // Ausgabe: 1337/10
    cout << b.zaehler << " / " << b.nenner << "\n"; // Ausgabe: 42/23
}
```

Regeln

Regeln

- C/C++ erlauben zusammengesetzte Datentypen
  - Anwendungszweck ganz ähnlich wie in Python
  - Aber: Struktur ist statisch
  - Felder müssen beim Übersetzen angegeben werden
- Zuweisungen kopieren den Inhalt
  - Achtung: Anders als in Python!
  - „Wert-Semantik“ der Typen in C++ („Value Types“)
  - `a = b` heißt: die Daten im Speicher von Variable `b` werden über den Speicher von Variable `a` „drüberkopiert“
    - (Inhalt von `a` wird überschrieben mit Inhalt von `b`)
  - Keine impliziten Referenzen! (alle Zeiger sind explizit)

Regeln für „structs“

Zugriff auf Felder

- Mit dem „`.`“ Operator, genau wie in Python
- Kann in beliebigen Ausdrücken verwendet werden
  - Als Wert/Operand in einer Berechnung
  - Als Ziel einer Zuweisen (linke Seite einer Zuweisung)
  - Typprüfung!

Andere Operatoren

- Alle anderen Operatoren (außer „=“) funktionieren erstmal nicht auf der ganzen Struktur (nicht definiert)
- Operatoren können dafür neu definiert („überladen“) werden (Details: siehe Lehrbuch/C++ Doku)

Das Alte Beispiel…

…mit den Brüchen

structs & classes

```
struct Bruch {  // es gibt auch „class“; das ist fast das gleiche; dazu mehr später
    int zaehler;
    int nenner;
}
void main() {
    Bruch a;
    int b = a;  // Fehler: Typ stimmt nicht! (bzw. kein Konvertierungs-Construktur definiert)
    Bruch c;
    a = a + c;  // Addition nicht erlaubt!
    // Abgesehen davon: nicht initialisiert; das stört C++ nicht, aber das Ergebnis wäre Zahlen-Müll.
}
```

Methoden (OOP)

structs & classes

```
struct Bruch {
   int zaehler;
   int nenner;
   // Konstruktoren heißen wie Klassen!
   // Default-Parameter wie in Python (alle Funktionen)
   Bruch(int z = 0, int n = 1) {
      zaehler = z; nenner = n; // kein „self“ nötig! (implizit)
   }
   void normalisieren() {
      int g = ggt(zaehler, nenner); // ggt anderswo definiert
      zaehler /= g; // kein „self“ nötig! (implizit)
      nenner /= g;
   }
}
```

Fancy-Fraction

structs & classes

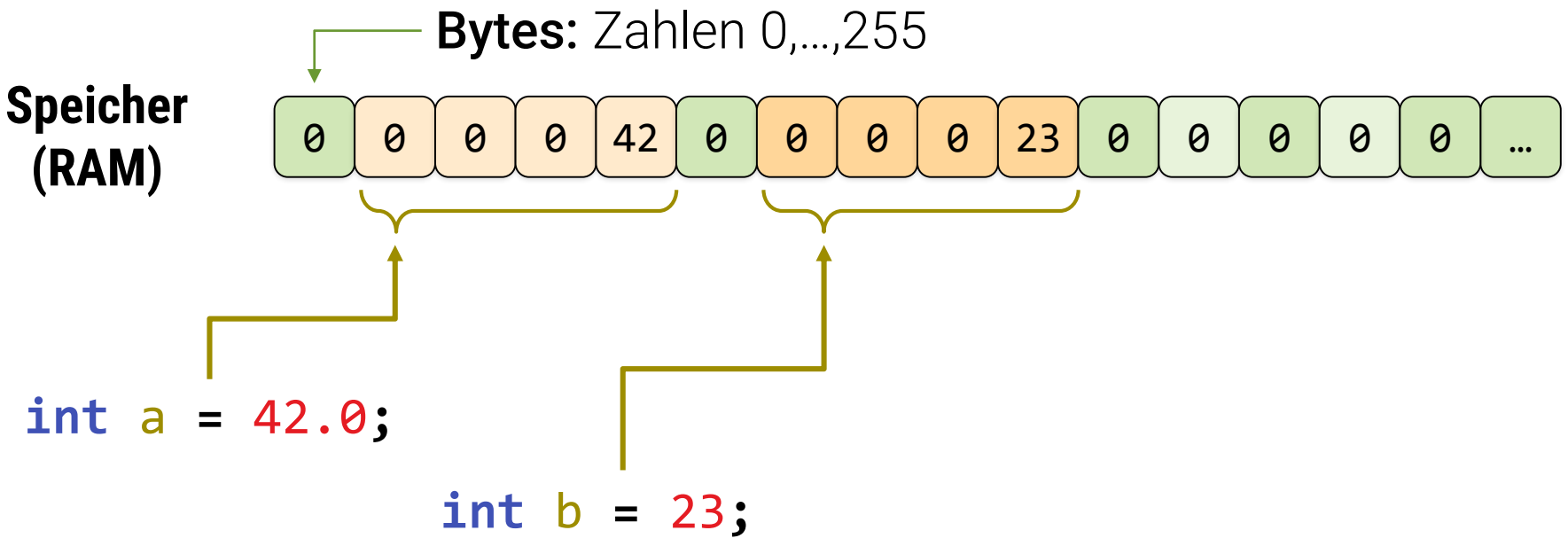
```
struct Bruch {
    int zaehler;
    int nenner;
    Bruch(int z = 0, int n = 1) {
        zaehler = z; nenner = n;
    }
    Bruch operator*(const Bruch &other) { // Operatoren überladen
        Bruch result;
        result.zaehler = zaehler * other.zaehler;
        result.nenner  = nenner  * other.nenner;
        return result;
    }
}
```

Fancy-Fraction

structs & classes

```
struct Bruch {
   …
   Bruch(int z = 0, int n = 0) {…}
   Bruch(const Bruch & other) { // „copy constructor“ für „Bruch b = …;“
      zaehler = other.zaehler;
      nenner = other.nenner;
   }
   void operator=(const Bruch &other) { // Zuweisungsoperator
      zaehler = other.zaehler;
      nenner = other.nenner;
   }
   bool operator==(const Bruch &other) { // Gleichheitsoperator
      return (double)zaehler/nenner == (double)other.zaehler/other.nenner;
   } // Achtung: sehr schlechte Implementierung (paßt noch auf Folie…)
}
```

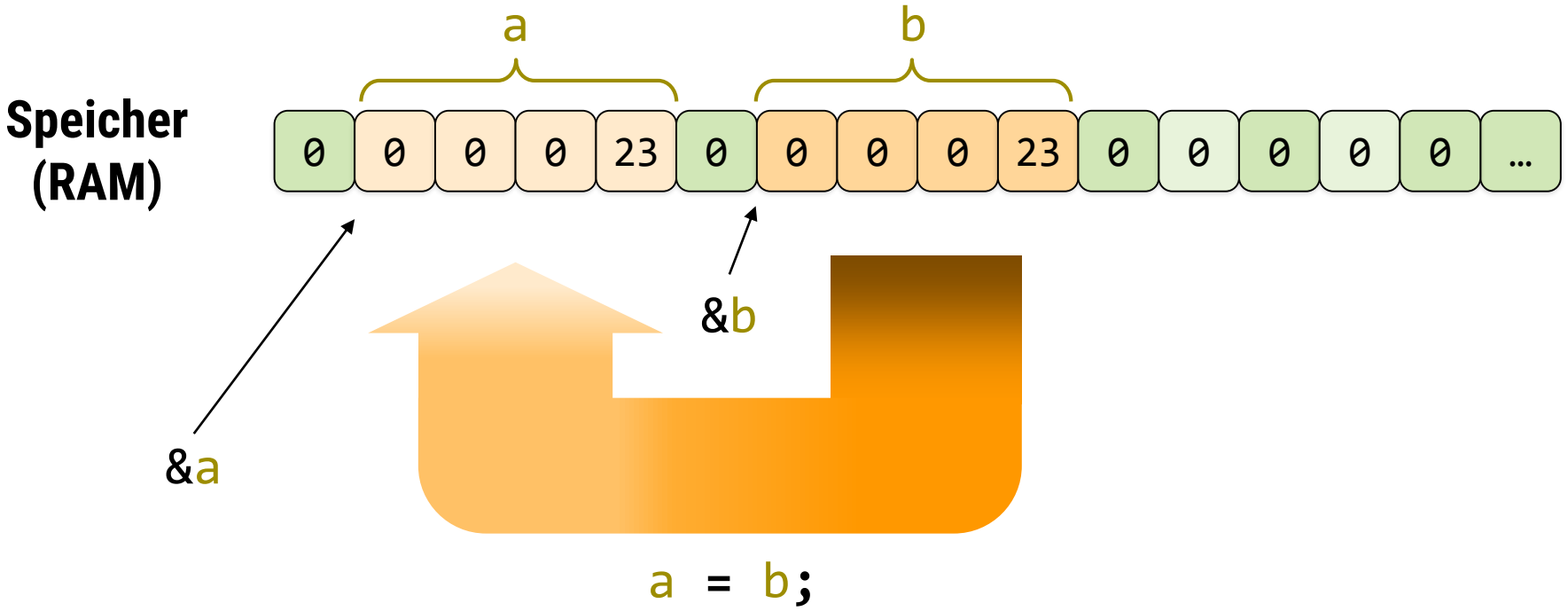
Maschinenrepräsentation



Größe von Datentypen

- `sizeof(<Typ>)` oder `sizeof(<Variable>)`
  - `size(int) == 4` (typischerweise)
  - `int32_t a; size(a) == 4`

Maschinenrepräsentation



Kopiere 4 Bytes („**`sizeof(int)`**“) ab der Speicherstelle,
in der a gespeichert ist („**&**a“) in den Speicher
von Speicherstelle b („**&**b“).

PS: So schnell wie möglich ;-)
Spar Dir alle Tests/Checks – der Compiler hat es erlaubt.

Zuweisungen & der ganze Rest

Zuweisungen kopieren Inhalte

- Gilt für einfache und zusammengesetzte Datentypen
- Gilt immer – Referenzen sind explizit
  - Eigene Datentypen

Wie bauen wir selbst Referenzen?