4. Zahlendarstellungen

Wertebereich der Typen **`int`**, **`float`** und **`double`** Gemischte Ausdrücke und Konversionen; Löcher im Wertebereich; Fliesskomma-Richtlinien;

Lernziele

Lernziele

- Sie haben ein gutes Verständnis dafür, wie mit dem Computer Zahlen repräsentiert werden.

Lernziele

- Sie haben ein gutes Verständnis dafür, wie mit dem Computer Zahlen repräsentiert werden.
- Sie können Ganzzahlen in **Binärdarstellung** bringen und damit rechnen.

Lernziele

- Sie haben ein gutes Verständnis dafür, wie mit dem Computer Zahlen repräsentiert werden.
- Sie können Ganzzahlen in **Binärdarstellung** bringen und damit rechnen.
- Sie verstehen, wie der Wertebereich von Ganzzahlen zustande kommt.

Lernziele

- Sie haben ein gutes Verständnis dafür, wie mit dem Computer Zahlen repräsentiert werden.
- Sie können Ganzzahlen in **Binärdarstellung** bringen und damit rechnen.
- Sie verstehen, wie der Wertebereich von Ganzzahlen zustande kommt.
- Sie können qualitativ die Repräsentation von Fliesskommazahlen beschreiben.

Lernziele

- Sie haben ein gutes Verständnis dafür, wie mit dem Computer Zahlen repräsentiert werden.
- Sie können Ganzzahlen in **Binärdarstellung** bringen und damit rechnen.
- Sie verstehen, wie der Wertebereich von Ganzzahlen zustande kommt.
- Sie können qualitativ die Repräsentation von Fliesskommazahlen beschreiben.
- Sie kennen die drei **Fliesskomma-Richtlinien**.

Lernziele

- Sie haben ein gutes Verständnis dafür, wie mit dem Computer Zahlen repräsentiert werden.
- Sie können Ganzzahlen in **Binärdarstellung** bringen und damit rechnen.
- Sie verstehen, wie der Wertebereich von Ganzzahlen zustande kommt.
- Sie können qualitativ die Repräsentation von Fliesskommazahlen beschreiben.
- Sie kennen die drei **Fliesskomma-Richtlinien**.
- Sie können mit Wahrheitswerten und **boolschen Ausdrücken** in Java umgehen.

Binäre Zahlendarstellungen

Binäre Darstellung (Bits aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$

Binäre Zahlendarstellungen

Binäre Darstellung (Bits aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Binäre Zahlendarstellungen

Binäre Darstellung (Bits aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

```
101011
```

Binäre Zahlendarstellungen

Binäre Darstellung (Bits aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

101011 entspricht **32+8+2+1**.

Binäre Zahlendarstellungen

Binäre Darstellung (Bits aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

101011 entspricht **43**.

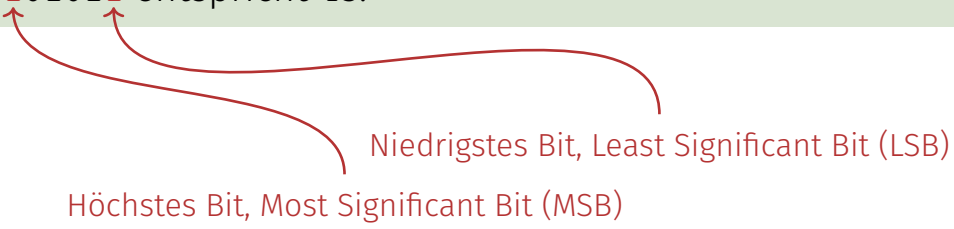
Binäre Zahlendarstellungen

Binäre Darstellung (Bits aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

101011 entspricht **43**.



Binäre Zahlen: Zahlen der Computer?

Wahrheit: Computer rechnen mit Binärzahlen.

NEUE ZÜRCHER ZEITUNG

TECHNIK

Mittwoch, 30. August 1950 Blatt 15
Mittagausgabe Nr. 1736 (30)

Das programmgesteuerte Rechengerät an der Eidgenössischen Technischen Hochschule in Zürich

Die Entwicklung programmgesteuerter Rechenmaschinen in den Vereinigten Staaten von Amerika wurde in den Artikeln „Elektronische Rechenmaschinen" (vgl. Nr. 2140 der „N. Z. Z." vom 13. Oktober 1948) und „Die neueste elektronische Rechenmaschine" (vgl. Nr. 571 der „N. Z. Z." vom 26. April 1950) behandelt. Nachstehend soll von einem Gerät deutscher Herkunft — Zuse K.-G., Neukirchen — die Rede sein, welches im Juli dieses Jahres am Institut für angewandte Mathematik der Eidgenössischen Technischen Hochschule in Zürich, das unter der Leitung von Prof. Dr. E. Stiefel steht, in Betrieb genommen wurde. Damit ist dieses Institut in der Lage, dem in der Schweiz immer stärker werdenden Bedürfnis nach einer leistungsfähigen Zentralstelle für numerische Rechnungen wenigstens teilweise gerecht zu werden. Bereits sind einige mathematische Probleme behandelt worden, und die Erledigung vieler anderer Aufgaben ist vorbereitet.

Merkmale des Gerätes

Das Gerät ist ein Glied in dem längeren Entwicklungsprogramm des Ingenieurs *Konrad Zuse*; es wurde im Auftrag des *Institutes für angewandte Mathematik* der E. T. H. unter Berücksichtigung von dessen Wünschen und Ideen von Zuse als *Modell Z 4*[1] konstruiert. Die ursprüngliche Entwicklung in Deutschland erfolgte in den Kriegsjahren und verlief völlig unabhängig von den Untersuchungen in den Vereinigten Staaten. Es ist überaus interessant festzustellen, wie für die meisten wichtigen funktionellen Probleme beiderorts genau dieselbe Lösung gefunden wurde, wie aber anderseits gewissen Fragen sekundärer Wichtigkeit eine ganz unterschiedliche Bedeutung beigemessen wurde.

Eine kurze technische Charakterisierung lautet wie folgt: Elektromechanisch arbeitendes Gerät mit 2200 Relais, 21 Schritteinheiten und einem Speicher für 64 Zahlen, welcher mit neuartigen, mechanischen Schaltgliedern arbeitet; Verwendung des Dualsystems und der halblogarithmischen Darstellung; Multiplikationszeit 2,5 Sekunden; Programmsteuerung mit Hilfe zweier Lochstreifen, auf die wahlweise umgeschaltet werden kann; Eingabe von Zahlen durch eine Tastatur oder durch einen Lochstreifen; Abgabe der Resultate durch Lampenfeld, Lochstreifen oder Druckwerk.

Das duale Zahlsystem

Lesen wir eine Dezimalzahl von rechts nach links, so erhöht sich das Gewicht von Stelle zu Stelle um den Faktor 10. Im *Dualsystem* ist nun einfach dieser Faktor 10 durch 2 zu ersetzen. Also bedeutet die (nunmehr duale) Zahl abcdefg den Ausdruck:

$$a \cdot 2^6 + b \cdot 2^5 + c \cdot 2^4 + d \cdot 2^3 + e \cdot 2^2 + f \cdot 2^1 + g \cdot 2^0$$

Die Zahl 1 wird in beiden Systemen gleich dargestellt. Um jedoch duale von dezimalen Zahlen deutlich zu trennen, schreiben wir die duale 1 als 1_2. Dagegen weicht schon die 2 ab, indem sie dual $L0$ lautet; denn dies bedeutet ja $1 \cdot 2^1 + 0 \cdot 2^0 = 2$. Wenn einer Zahl (ohne Stellen nach dem Komma) rechts eine Null zugefügt wird, so verändert sie sich um den Faktor 2 (und nicht, wie im Dezimalsystem, um den Faktor 10). Auf diese Weise kann aus $L0 = 2$ auf einfachste Weise gebildet werden: $L00 = 4$, $L000 = 8$, $L0000 = 16$, usw.

Die Dualzahl $LOL0L$ bedeutet nun also:

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 21$$

Ganz analog sind etwaige Stellen nach dem Komma zu interpretieren; so wird $L,\ 0L1$ wie folgt übersetzt:

$$1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 1 + 1/4 + 1/8 = 1{,}375$$

Aenderung des Maßstabes

durchgerechnet werden können.

Die beschriebene Darstellung bringt eine gewisse Komplikation der Rechenoperationen mit sich. So müssen vor einer Addition die beiden Summanden zunächst so verschoben werden, daß ihre Kommata untereinander zu liegen kommen, was an Hand eines Beispiels erläutert werden soll. Damit der Leser nicht durch das ungewohnte duale Zahlsystem verwirrt wird, ist das Beispiel im Dezimalsystem durchgeführt; doch wird daran erinnert, daß das Gerät in Wirklichkeit mit dualen Zahlen rechnet.

Es soll also etwa addiert werden: $2{,}345078 \times 10^5 + 9{,}876513 \times 10^{-1}$ (Man beachte, daß die eigentliche Zahl stets zwischen 1 und 10 liegt, also das Komma nach der ersten Stelle hat.) Nun müssen die beiden Summanden „ausgerichtet" werden, d. h. die beiden Exponenten sind einander gleich zu machen, und zwar erhält der kleinere Exponent den Wert des größeren, also 2. Die Zahlen lauten nun, richtig untereinander geschrieben und addiert, wie folgt:

$$\begin{array}{r} 2{,}345078 \times 10^5 \\ 0{,}009876 \times 10^5 \\ \hline 2{,}354554 \times 10^5 \end{array}$$

Es ist ersichtlich, daß bei der kleineren der beiden

Abb. 2. Der Schaltpult bei der Fertigung eines Rechenrahmes. Die Ablester für den Loch-

Binäre Zahlen: Zahlen der Computer?

Klischee: Computer reden 0/1-Kauderwelsch.

Binäre Binäre Binäre

01001110 01011010 01011010

Freitag, 8. Juni 2012 · Nr. 131 · 233. Jhg. 01001010 01010110 01001101 www.nzz.ch · Fr. 4.00 · €3.50

01000010 01100101
01110010 01101001

01100011 01101000 01110100 01100101

00100000 11111100 01100010
01100001 01110010 00100000
01101110 01100101 01110101 011-
00101 01110011 00100000 010-
01101 01100001 01110011

01110011 01100001 01100010 00100000 011-
01001 01101110 00100000 01010011 01111-
001 01110010 01101001 01100101 01101110
00001101 00001010 00001101 00000010
01010101 01101110 01101111 00101101 010-
00100 01100101 01001111 01100000 01100001
01100011 01100000 01110100 01100001 011-
10010 00100000 01110110 01001111 01001101
00100000 01010011 01100011 01101000 011-
00001 01100101 01110000 01101100 01000-
001 01101000 01111010 00100000

01100110 01100101

01110010 01101110 01100111 01100101 011-

01100001 01110011 00100000 01100101 011-
00001 01110011 01100011 01100001 01101-
011 01100101 01110010 00100000 01110011
01100100 01100001 01110000 01110000 011-
00111 01000101 01101110 01101101 01101110
01100100 01100001 01101110 00101100 001-
00000 01000100 01100001 01000101 001-
00000 01010010 01100001 01100111 01101001
01100101 01110030 01101001 01101110
01100011 00100000 01101101 01100001 011-
00011 01101000 01110100 01100101 00100-
000 01100101 01101110 01100111 01100101
01101110 01100001 01101110 01101110 011-
01000 01100101 00100000 10101011 01001000
01100101 01110000 01101110 01101111 011-
10000 01101001 01100011 01110000 01100001
01101110 10111001 00100000 01100100 011-
00001 01100110 11111100 01110010

00100000 01110110

01100101 01110010 01100001 01101110 011-
0100 01110111 01101111 01110010 01110100
01101100 01101001 01000011 01101000 001-

Definition: Wertebereich

Bei numerischen Typen gibt der Wertebereich gibt an, welches Zahlenintervall abgedeckt werden kann.

Buch auf Seite 24

Wertebereich des Typs `int`

```
public class Main {
    public static void main(String[] args) {
        Out.print("Minimum int value is ");
        Out.println(Integer.MIN_VALUE);
        Out.print("Maximum int value is ");
        Out.println(Integer.MAX_VALUE);
    }
}
```

Wertebereich des Typs `int`

```
public class Main {
    public static void main(String[] args) {
        Out.print("Minimum int value is ");
        Out.println(Integer.MIN_VALUE);
        Out.print("Maximum int value is ");
        Out.println(Integer.MAX_VALUE);
    }
}
```

```
Minimum int value is -2147483648.
Maximum int value is 2147483647.
```

Wertebereich des Typs `int`

```
public class Main {
    public static void main(String[] args) {
        Out.print("Minimum int value is ");
        Out.println(Integer.MIN_VALUE);
        Out.print("Maximum int value is ");
        Out.println(Integer.MAX_VALUE);
    }
}
```

```
Minimum int value is -2147483648.
Maximum int value is 2147483647.
```

Woher kommen diese Zahlen?

Wertebereich des Typs `int`

Repräsentation mit 32 Bits. Wertebereich

$$\{-2^{31}, \ldots, -1, 0, 1, \ldots, 2^{31} - 2, 2^{31} - 1\}$$

Negative Zahlen (3 Stellen)

	a	$-a$
0	000	
1	001	
2	010	
3	011	
4	100	
5	101	
6	110	
7	111	

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001		
2	010		
3	011		
4	100		
5	101		
6	110		
7	111		

Das höchste Bit entscheidet über das Vorzeichen.

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	**111**	-1
2	010		
3	011		
4	100		
5	101		
6	110		
7	**111**		

Das höchste Bit entscheidet über das Vorzeichen.

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	**110**	-2
3	011		
4	100		
5	101		
6	**110**		
7	111		

Das höchste Bit entscheidet über das Vorzeichen.

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	**101**	-3
4	100		
5	**101**		
6	110		
7	111		

Das höchste Bit entscheidet über das Vorzeichen.

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	101	-3
4	**100**	**100**	-4
5	101		
6	110		
7	111		

Das höchste Bit entscheidet über das Vorzeichen.

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	101	-3
4	100	100	-4
5	101		
6	110		
7	111		

Das höchste Bit entscheidet über das Vorzeichen.

Überlauf und Unterlauf

- Arithmetische Operationen (`+`,`-`,`*`) können aus dem Wertebereich herausführen.
- Ergebnisse können inkorrekt sein.

`power8`: $15^8 = -1732076671$ **`power20`**: $3^{20} = -808182895$

- Es gibt **keine Fehlermeldung!**

Definition: Fliesskommazahlen

*Fliesskommazahlen stellen Zahlen aus $\mathbb{R}$ dar mit einer festen Anzahl **signifikanter** Stellen, multipliziert mit einer Zehnerpotenz. (Basis 10).*

Buch auf Seite 67

„Richtig Rechnen”

```
public class Main {
    public static void main(String[] args) {
        Out.print("Celsius: ");
        int celsius = In.readInt();
        int fahrenheit = 9 * celsius / 5 + 32;
        Out.print(celsius + " degrees Celsius are ");
        Out.println(fahrenheit + " degrees Fahrenheit");
    }
}
```

```
28 degrees Celsius are 82 degrees Fahrenheit.
```

„Richtig Rechnen”

```
public class Main {
    public static void main(String[] args) {
        Out.print("Celsius: ");
        int celsius = In.readInt();
        int fahrenheit = 9 * celsius / 5 + 32;
        Out.print(celsius + " degrees Celsius are ");
        Out.println(fahrenheit + " degrees Fahrenheit");
    }
}
```

```
28 degrees Celsius are 82 degrees Fahrenheit.
```

richtig wäre 82.4

„Richtig Rechnen”

```
public class Main {
    public static void main(String[] args) {
        Out.print("Celsius: ");
        float celsius = In.readInt();
        float fahrenheit = 9 * celsius / 5 + 32;
        Out.print(celsius + " degrees Celsius are ");
        Out.println(fahrenheit + " degrees Fahrenheit");
    }
}
```

28 degrees Celsius are 82.4 degrees Fahrenheit.

Typen `float` und `double`

- sind die fundamentalen Typen für Fliesskommazahlen
- approximieren den Körper der reellen Zahlen $(\mathbb{R}, +, \times)$ in der Mathematik
- haben grossen Wertebereich, ausreichend für viele Anwendungen (**`double`** hat mehr Stellen als **`float`**)
- sind auf vielen Rechnern sehr schnell

Fixkommazahlen

- feste Anzahl Vorkommastellen (z.B. 7)
- feste Anzahl Nachkommastellen (z.B. 3)

Fixkommazahlen

- feste Anzahl Vorkommastellen (z.B. 7)
- feste Anzahl Nachkommastellen (z.B. 3)

```
82.4 = 0000082.400
```

Fixkommazahlen

- feste Anzahl Vorkommastellen (z.B. 7)
- feste Anzahl Nachkommastellen (z.B. 3)

```
82.4 = 0000082.400
```

Nachteile

- Wertebereich wird *noch* kleiner als bei ganzen Zahlen.
- Repräsentierbarkeit hängt von der Stelle des Kommas ab.

Fixkommazahlen

- feste Anzahl Vorkommastellen (z.B. 7)
- feste Anzahl Nachkommastellen (z.B. 3)

```
0.0824 = 0000000.082
```
← dritte Stelle abgeschnitten

Nachteile

- Wertebereich wird *noch* kleiner als bei ganzen Zahlen.
- Repräsentierbarkeit hängt von der Stelle des Kommas ab.

Fliesskommazahlen

- feste Anzahl signifikante Stellen (z.B. 10)
- plus Position des Kommas

Fliesskommazahlen

- feste Anzahl signifikante Stellen (z.B. 10)
- plus Position des Kommas

`82.4` = $824 \cdot 10^{-1}$
`0.0824` = $824 \cdot 10^{-4}$

Fliesskommazahlen

- feste Anzahl signifikante Stellen (z.B. 10)
- plus Position des Kommas

`82.4` = $824 \cdot 10^{-1}$
`0.0824` = $824 \cdot 10^{-4}$

- Zahl ist $\mathit{Signifikand} \times 10^{\mathit{Exponent}}$

Wertebereich

Ganzzahlige Typen:

- Über- und Unterlauf häufig, aber …

Wertebereich

Ganzzahlige Typen:

- Über- und Unterlauf häufig, aber …
- Wertebereich ist zusammenhängend (keine „Löcher”): $\mathbb{Z}$ ist „diskret”.

Wertebereich

Ganzzahlige Typen:

- Über- und Unterlauf häufig, aber ...
- Wertebereich ist zusammenhängend (keine „Löcher”): $\mathbb{Z}$ ist „diskret”.

Fliesskommatypen:

- Über- und Unterlauf selten, aber ...

Wertebereich

Ganzzahlige Typen:

- Über- und Unterlauf häufig, aber ...
- Wertebereich ist zusammenhängend (keine „Löcher”): $\mathbb{Z}$ ist „diskret”.

Fliesskommatypen:

- Über- und Unterlauf selten, aber ...

Wertebereich

Ganzzahlige Typen:

- Über- und Unterlauf häufig, aber …
- Wertebereich ist zusammenhängend (keine „Löcher”): $\mathbb{Z}$ ist „diskret”.

Fliesskommatypen:

- Über- und Unterlauf selten, aber …
- es gibt Löcher: $\mathbb{R}$ ist „kontinuierlich”.

Löcher im Wertebereich

```
public class Main {
    public static void main(String[] args) {
    Out.print("First number =? ");
    float n1 = In.readFloat();

    Out.print("Second number =? ");
    float n2 = In.readFloat();

    Out.print("Their difference =? ");
    float d = In.readFloat();

    Out.print("computed difference - input difference = ");
    Out.println(n1-n2-d);
    }
}
```

Löcher im Wertebereich

```
public class Main {
    public static void main(String[] args) {
    Out.print("First number =? ");
    float n1 = In.readFloat();

    Out.print("Second number =? ");
    float n2 = In.readFloat();

    Out.print("Their difference =? ");
    float d = In.readFloat();

    Out.print("computed difference - input difference = ");
    Out.println(n1-n2-d);
    }
}
```

Eingabe 1.5

Eingabe 1.0

Eingabe 0.5

Löcher im Wertebereich

```
public class Main {
    public static void main(String[] args) {
    Out.print("First number =? ");
    float n1 = In.readFloat();

    Out.print("Second number =? ");
    float n2 = In.readFloat();

    Out.print("Their difference =? ");
    float d = In.readFloat();

    Out.print("computed difference - input difference = ");
    Out.println(n1-n2-d);
    }
}
```

Eingabe 1.5

Eingabe 1.0

Eingabe 0.5

Ausgabe 0

Löcher im Wertebereich

```
public class Main {
    public static void main(String[] args) {
    Out.print("First number =? ");
    float n1 = In.readFloat();

    Out.print("Second number =? ");
    float n2 = In.readFloat();

    Out.print("Their difference =? ");
    float d = In.readFloat();

    Out.print("computed difference - input difference = ");
    Out.println(n1-n2-d);
    }
}
```

Eingabe 1.1

Eingabe 1.0

Eingabe 0.1

Löcher im Wertebereich

```
public class Main {
    public static void main(String[] args) {
    Out.print("First number =? ");
    float n1 = In.readFloat();

    Out.print("Second number =? ");
    float n2 = In.readFloat();

    Out.print("Their difference =? ");
    float d = In.readFloat();

    Out.print("computed difference - input difference = ");
    Out.println(n1-n2-d);
    }
}
```

Eingabe 1.1

Eingabe 1.0

Eingabe 0.1

Ausgabe 2.2351742E-8

Löcher im Wertebereich

```
public class Main {
    public static void main(String[] args) {
    Out.print("First number =? ");
    float n1 = In.readFloat();

    Out.print("Second number =? ");
    float n2 = In.readFloat();

    Out.print("Their difference =? ");
    float d = In.readFloat();

    Out.print("computed difference - input difference = ");
    Out.println(n1-n2-d);
    }
}
```

Eingabe 1.1

Eingabe 1.0

Eingabe 0.1

Ausgabe 2.2351742E-8

Ja was ist denn hier los?

Regel 1

Teste keine gerundeten Fliesskommazahlen auf Gleichheit!

Regel 1

Teste keine gerundeten Fliesskommazahlen auf Gleichheit!

```
for (float i = 0.1; i != 1.0; i += 0.1){ // doesn't work!!!
    Out.println(i);
}
```

Regel 1

Teste keine gerundeten Fliesskommazahlen auf Gleichheit!

```
for (float i = 0.1; i != 1.0; i += 0.1){ // doesn't work!!!
    Out.println(i);
}
```

Endlosschleife, weil i niemals exakt 1 ist!

Mehr dazu nächstes mal!

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

Angenommen wir rechnen mit 4 Stellen Genauigkeit

$$
\begin{aligned}
& 1.000 \cdot 10^5 \\
+ & 1.000 \cdot 10^0 \\
= & \ 1.00001 \cdot 10^5 \\
\text{“=”} & \ 1.000 \cdot 10^5 \quad \text{(Rundung auf 4 Stellen)}
\end{aligned}
$$

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

Angenommen wir rechnen mit 4 Stellen Genauigkeit

$$
\begin{aligned}
& 1.000 \cdot 10^5 \\
+ & 1.000 \cdot 10^0 \\
= & \ 1.00001 \cdot 10^5 \\
\text{“=”} & \ 1.000 \cdot 10^5 \quad \text{(Rundung auf 4 Stellen)}
\end{aligned}
$$

Addition von 1 hat keinen Effekt!

Regel 3

Subtrahiere keine zwei Zahlen sehr ähnlicher Grösse!

Auslöschungsproblematik (ohne weitere Erklärung).

5. Wahrheitswerte

Boolesche Funktionen; der Typ **`boolean`**; logische und relationale Operatoren; Kurzschlussauswertung

Wo wollen wir hin?

```
int a = In.readInt();
if (a % 2 == 0) {
    Out.print("even");
} else {
    Out.print("odd");
}
```

Wo wollen wir hin?

```
int a = In.readInt();
if (a % 2 == 0) {
    Out.print("even");
} else {
    Out.print("odd");
}
```

Verhalten hängt ab vom Wert eines **Boolschen Ausdrucks**

Boolesche Werte in der Mathematik

Boolesche Ausdrücke können zwei mögliche Werte annehmen:

wahr oder **falsch**

Der Typ `boolean` in Java

- Repräsentiert **Wahrheitswerte**

Der Typ `boolean` in Java

- Repräsentiert **Wahrheitswerte**
- Literale `true` und `false`

Der Typ `boolean` in Java

- Repräsentiert **Wahrheitswerte**
- Literale `true` und `false`
- Wertebereich {***true***, ***false***}

```
boolean b = true; //Variable b with value true
```

Relationale Operatoren

`a < b` (kleiner als)

Zahlentyp $\times$ Zahlentyp $\rightarrow$ `boolean`

Relationale Operatoren

`a < b` (kleiner als)

```
boolean b = (1 < 3); // b =
```

Relationale Operatoren

`a < b` (kleiner als)

```
boolean b = (1 < 3); // b = true (wahr)
```

Relationale Operatoren

`a >= b` (grösser gleich)

```
int a = 0;
boolean b = (a >= 3); // b =
```

Relationale Operatoren

`a >= b` (grösser gleich)

```
int a = 0;
boolean b = (a >= 3); // b = false (falsch)
```

Relationale Operatoren

`a == b` (gleich)

```
int a = 4;
boolean b = (a % 3 == 1); // b =
```

Relationale Operatoren

`a == b` (gleich)

```
int a = 4;
boolean b = (a % 3 == 1); // b = true (wahr)
```

Relationale Operatoren

`a != b` (ungleich)

```
int a = 1;
boolean b = (a != 2*a-1); // b =
```

Relationale Operatoren

`a != b` (ungleich)

```
int a = 1;
boolean b = (a != 2*a-1); // b = false (falsch)
```

Boolesche Funktionen in der Mathematik

- Boolesche Funktion

$$f : \{0,1\}^2 \to \{0,1\}$$

- 0 entspricht „falsch“.
- 1 entspricht „wahr“.

$\mathrm{AND}(x, y)$ $x \wedge y$

- „Logisches Und“

$$f : \{0,1\}^2 \to \{0,1\}$$

- 0 entspricht „falsch“.
- 1 entspricht „wahr“.

x	y	$\mathrm{AND}(x, y)$
0	0	0
0	1	0
1	0	0
1	1	1

Logischer Operator &&

a && b (logisches Und)

```
int n = -1;
int p = 3;
boolean b = (n < 0) && (0 < p); //
```

Logischer Operator &&

a && b (logisches Und)

```
int n = -1;
int p = 3;
boolean b = (n < 0) && (0 < p); // b = true (wahr)
```

$\mathrm{OR}(x, y)$ $x \vee y$

- „Logisches Oder“

$$f : \{0, 1\}^2 \to \{0, 1\}$$

- 0 entspricht „falsch“.
- 1 entspricht „wahr“.

x	y	$\mathrm{OR}(x, y)$
0	0	0
0	1	1
1	0	1
1	1	1

Logischer Operator ||

`a || b` (logisches Oder)

```
int n = 1;
int p = 0;
boolean b = (n < 0) || (0 < p); //
```

Logischer Operator ||

`a || b` (logisches Oder)

```
int n = 1;
int p = 0;
boolean b = (n < 0) || (0 < p); // b = false (falsch)
```

$\mathrm{NOT}(x)$ $\neg x$

- „Logisches Nicht“

$$f : \{0, 1\} \to \{0, 1\}$$

- 0 entspricht „falsch“.
- 1entspricht „wahr“.

x	$\mathrm{NOT}(x)$
0	1
1	0

Logischer Operator !

`!b` (logisches Nicht)

$$\texttt{boolean} \rightarrow \texttt{boolean}$$

Logischer Operator !

`!b` (logisches Nicht)

```
int n = 1;
boolean b = !(n < 0); //
```

Logischer Operator !

`!b` (logisches Nicht)

```
int n = 1;
boolean b = !(n < 0); // b = true (wahr)
```

Präzedenzen

```
!b && a
```

Präzedenzen

```
!b && a
```

```
(!b) && a
```

Präzedenzen

```
a && b || c && d
```

Präzedenzen

```
a && b || c && d
```

```
(a && b) || (c && d)
```

Präzedenzen

```
a || b   &&   c || d
```

Präzedenzen

```
a || b   &&   c || d
```

```
a || (b && c) || d
```

Präzedenzen

```
7 + x < y && y != 3 * z || ! b
```

Präzedenzen

Der unäre logische Operator `!`
bindet stärker als

```
7 + x < y && y != 3 * z || (!b)
```

Präzedenzen

Der unäre logische Operator `!`
bindet stärker als
binäre arithmetische Operatoren. Diese
binden stärker als

```
(7 + x) < y && y != (3 * z) || (!b)
```

Präzedenzen

Der unäre logische Operator `!`
bindet stärker als
binäre arithmetische Operatoren. Diese
binden stärker als
relationale Operatoren,
und diese binden stärker als

```
((7 + x) < y) && (y != (3 * z)) || (!b)
```

Präzedenzen

Der unäre logische Operator `!`
bindet stärker als
binäre arithmetische Operatoren. Diese
binden stärker als
relationale Operatoren,
und diese binden stärker als
binäre logische Operatoren.

```
((7 + x) < y) && (y != (3 * z)) || (!b)
```

Einige Klammern auf den vorher gezeigten Folien waren unnötig.

DeMorgansche Regeln

- `!(a && b) == (!a || !b)`

DeMorgansche Regeln

- `!(a && b) == (!a || !b)`

`!` (reich *und* schön) == (arm *oder* hässlich)

DeMorgansche Regeln

- `!(a && b) == (!a || !b)`
- `!(a || b) == (!a && !b)`

`!` (reich *und* schön) == (arm *oder* hässlich)

Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

Kurzschlussauswertung

- Logische Operatoren **`&&`** und **`||`** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 6 $\Rightarrow$

```
x != 0 && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren **`&&`** und **`||`** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 6 $\Rightarrow$

```
true && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren **&&** und **||** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 6 $\Rightarrow$

```
true && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren **`&&`** und **`||`** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 0 $\Rightarrow$

```
x != 0 && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren **`&&`** und **`||`** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 0 $\Rightarrow$

```
false && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren **`&&`** und **`||`** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 0 $\Rightarrow$ ***`false`*** **`(falsch)`**

Kurzschlussauswertung

- Logische Operatoren **`&&`** und **`||`** werten den *linken Operanden zuerst* aus.
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet.

x hat Wert 0 $\Rightarrow$

```
x != 0 && z / x > y
```

$\Rightarrow$ Keine Division durch 0