4. Zahlendarstellungen

Wertebereich der Typen `int`, `float` und `double` Gemischte Ausdrücke und Konversionen; Löcher im Wertebereich; Fliesskommazahlensysteme; IEEE Standard; Grenzen der Fliesskommaarithmetik; Fliesskomma-Richtlinien;

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 $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Beispiel: `101011`

Binäre Zahlendarstellungen

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

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

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

Beispiel: `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 $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Beispiel: `101011` entspricht `43`.

Binäre Zahlendarstellungen

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

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

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

Beispiel: 101011 entspricht 43.

Binäre Zahlen: Zahlen der Computer?

Wahrheit: Computer rechnen mit Binärzahlen.

NEUE ZÜRCHER ZEITUNG

TECHNIK

Mittwoch, 26. August 1953 Blatt 3
Mittagausgabe Nr. 1734 (59)

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. 871 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 weiterer 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"* konstruiert. Die ursprüngliche Entwicklung in Deutschland erfolgte in den Kriegsjahren und verlief völlig unabhängig von den Unternehmungen in den Vereinigten Staaten. Es ist übrigens interessant festzustellen, wie für die meisten wichtigen funktionellen Probleme beiderseits 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 Schreibschaltern 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

Allgemein wird programmgesteuerten Rechengeräten häufig das duale Zahlsystem zugrunde gelegt, welches nur die zwei Zahlzeichen 0 und 1 verwendet, während das bekannte Dezimalsystem

Lesen wir eine Dezimalzahl von rechts nach links, so erhält 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_D. — Dagegen weicht schon die 2 ab, indem sie dual 10_D 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 vergrößert sie sich um den Faktor 2 (und nicht, wie im Dezimalsystem, um den Faktor 10). Auf diese Weise kann aus $1_D 0 = 2$ auf einfachste Weise gebildet werden: $1_D 00 = 4$, $1_D 000 = 8$, $1_D 0000 = 16$, usw.

Die Dualzahl $1_D 01_D 01_D$ 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 $1_D 01_D$ wie folgt übersetzt:

$$1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 1 + {}^1/_4 = {}^5/_4 = 1{,}375$$

Der große Vorteil, der das Dualsystem für Rechenmaschinen so geeignet macht, nämlich die Reduktion der Anzahl der verwendeten Symbole auf nur zwei, wird durch den großen einen Nachteil erkauft: Es braucht mehr Stellen, um eine bestimmte Zahl darzustellen. Die einstellige Zahl 9

Änderung 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 darum ersucht, daß das Gerät in Wirklichkeit mit dualen Zahlen rechnet.

Es soll also etwa addiert werden: $2{,}345678 \times 10^5 + 9{,}876543 \times 10^1$ (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:

$$\begin{array}{l} 2{,}345678 \times 10^5 \\ 0{,}009876 \times 10^5 \\ \hline 2{,}355554 \times 10^5 \end{array}$$

Es ist ersichtlich, daß bei der kleineren der beiden Zahlen rechts einige Stellen abgeschnitten werden müssen; dann wenn die Summanden siebenstellig angegeben waren, so soll auch das Resultat nicht mehr als sieben Stellen enthalten.

Abb. 2. Der Schaltpult bei der Fertigung eines Rechenpriest. Die Abtaster für den Lochstreifen sind deutlich sichtbar.

Befehle können „bedingt" gegeben werden, d. h. ihre Ausführung wird von der Natur eines errechneten Resultates abhängig gemacht. Erst dadurch werden die mathematischen Probleme

Binäre Zahlen: Zahlen der Computer?

Klischee: Computer reden 0/1-Kauderwelsch.

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

01100110 01101100 11111100

01101000

01100110 01101101

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

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

Wertebereich des Typs `int`

Repräsentation mit 32 Bits. Wertebereich

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

Woher kommt gerade diese Aufteilung?

Rechnen mit Binärzahlen (4 Stellen)

Einfache Addition

$$\begin{array}{r} 2 \\ +3 \\ \hline 5 \end{array} \qquad \begin{array}{r} 0010 \\ +0011 \\ \hline 0101 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Einfache Subtraktion

$$\begin{array}{r} 5 \\ -3 \\ \hline 2 \end{array} \qquad \begin{array}{r} 0101 \\ -0011 \\ \hline 0010 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Addition mit Überlauf

$$\begin{array}{r} 7 \\ +9 \\ \hline 16 \end{array} \qquad \begin{array}{r} 0111 \\ +1001 \\ \hline (1)0000 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Negative Zahlen?

$$\begin{array}{r} 5 \\ +(-5) \\ \hline 0 \end{array} \qquad \begin{array}{r} 0101 \\ ???? \\ \hline (1)0000 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Einfacher: -1

$$\begin{array}{r} 1 \\ +(-1) \\ \hline 0 \end{array} \qquad \begin{array}{r} 0001 \\ 1111 \\ \hline (1)0000 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Nutzen das aus:

$$\begin{array}{r} 3 \\ +? \\ \hline -1 \end{array} \qquad \begin{array}{r} 0011 \\ +???? \\ \hline 1111 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Invertieren!

$$\begin{array}{r} 3 \\ +(-4) \\ \hline -1 \end{array} \qquad \begin{array}{r} 0011 \\ +1100 \\ \hline 1111 \end{array} \widehat{=} 2^B - 1$$

Rechnen mit Binärzahlen (4 Stellen)

$$\begin{array}{r} a \\ +(-a-1) \\ \hline -1 \end{array} \qquad \begin{array}{r} a \\ \bar{a} \\ \hline 1111 \end{array} \widehat{=} 2^B - 1$$

Rechnen mit Binärzahlen (4 Stellen)

- Negation: Inversion und Addition von 1

$$-a \quad \widehat{=} \quad \bar{a} + 1$$

Rechnen mit Binärzahlen (4 Stellen)

- Wrap-around Semantik (Rechnen modulo 2^B)

$$-a \quad \widehat{=} \quad 2^B - a$$

Warum das funktioniert

Modulo-Arithmetik: Rechnen im Kreis[4]

[4] Die Arithmetik funktioniert auch mit Dezimalzahlen (und auch für die Multiplikation)

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		

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**		

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		

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		

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		

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		

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!*

„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

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

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

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.

Fixkommazahlen

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

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

Nachteile

- Repräsentierbarkeit hängt von der Stelle des Kommas ab.

Fliesskommazahlen

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

$$\texttt{82.4} = 824 \cdot 10^{-1}$$

$$\texttt{0.0824} = 824 \cdot 10^{-4}$$

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

Fliesskommazahlen

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

$$\texttt{82.4} = 824 \cdot 10^{-1}$$

$$\texttt{0.0824} = 824 \cdot 10^{-4}$$

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

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 *Signifikand* $\times 10^{\textit{Exponent}}$

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

Fliesskommatypen:

- Über- und Unterlauf selten, aber ...

Wertebereich

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?

Fliesskommazahlensysteme

Ein Fliesskommazahlensystem ist durch vier natürliche Zahlen definiert:

- $\beta \geq 2$, die Basis,

Fliesskommazahlensysteme

Ein Fliesskommazahlensystem ist durch vier natürliche Zahlen definiert:

- $\beta \geq 2$, die Basis,
- $p \geq 1$, die Präzision (Stellenzahl),

Fliesskommazahlensysteme

Ein Fliesskommazahlensystem ist durch vier natürliche Zahlen definiert:

- $\beta \geq 2$, die Basis,
- $p \geq 1$, die Präzision (Stellenzahl),
- $e_{\min}$, der kleinste Exponent,

Fliesskommazahlensysteme

Ein Fliesskommazahlensystem ist durch vier natürliche Zahlen definiert:

- $\beta \geq 2$, die Basis,
- $p \geq 1$, die Präzision (Stellenzahl),
- $e_{\min}$, der kleinste Exponent,
- $e_{\max}$, der grösste Exponent.

Fliesskommazahlensysteme

Ein Fliesskommazahlensystem ist durch vier natürliche Zahlen definiert:

- $\beta \geq 2$, die Basis,
- $p \geq 1$, die Präzision (Stellenzahl),
- $e_{\min}$, der kleinste Exponent,
- $e_{\max}$, der grösste Exponent.

Bezeichnung:

$$F(\beta, p, e_{\min}, e_{\max})$$

Fliesskommazahlensysteme

$F(\beta, p, e_{\min}, e_{\max})$ enthält die Zahlen

$$\pm \sum_{i=0}^{p-1} d_i \beta^{-i} \cdot \beta^e,$$

$d_i \in \{0, \ldots, \beta - 1\}, \quad e \in \{e_{\min}, \ldots, e_{\max}\}.$

Fliesskommazahlensysteme

$F(\beta, p, e_{\min}, e_{\max})$ enthält die Zahlen

$$\pm \sum_{i=0}^{p-1} d_i \beta^{-i} \cdot \beta^e,$$

$$d_i \in \{0, \ldots, \beta - 1\}, \quad e \in \{e_{\min}, \ldots, e_{\max}\}.$$

In Basis-β-Darstellung:

$$\pm\, d_0 \bullet d_1 \ldots d_{p-1} \times \beta^e,$$

Fliesskommazahlensysteme

Beispiel

- $\beta = 10$

Darstellungen der Dezimalzahl 0.1

$$1.0 \cdot 10^{-1}, \quad 0.1 \cdot 10^{0}, \quad 0.01 \cdot 10^{1}, \quad \ldots$$

Normalisierte Darstellung

Normalisierte Zahl:

$$\pm\, d_0 \bullet d_1 \ldots d_{p-1} \times \beta^e, \qquad d_0 \neq 0$$

Bemerkung 1

Die normalisierte Darstellung ist eindeutig und deshalb zu bevorzugen.

Normalisierte Darstellung

Normalisierte Zahl:

$$\pm\, d_0 \bullet d_1 \ldots d_{p-1} \times \beta^e, \qquad d_0 \neq 0$$

Bemerkung 1

Die normalisierte Darstellung ist eindeutig und deshalb zu bevorzugen.

Normalisierte Darstellung

Normalisierte Zahl:

$$\pm\, d_0 \bullet d_1 \ldots d_{p-1} \times \beta^e, \qquad d_0 \neq 0$$

Bemerkung 2

Die Zahl 0 (und alle Zahlen kleiner als $\beta^{e_{\min}}$) haben keine normalisierte Darstellung (werden wir später beheben)!

Menge der normalisierten Zahlen

$$F^*(\beta, p, e_{\min}, e_{\max})$$

Normalisierte Darstellung

Beispiel $F^*(2, 3, -2, 2)$ (nur positive Zahlen)

$d_0 \bullet d_1 d_2$	$e = -2$	$e = -1$	$e = 0$	$e = 1$	$e = 2$
1.00_2	0.25	0.5	1	2	4
1.01_2	0.3125	0.625	1.25	2.5	5
1.10_2	0.375	0.75	1.5	3	6
1.11_2	0.4375	0.875	1.75	3.5	7

Normalisierte Darstellung

Beispiel $F^*(2, 3, -2, 2)$ (nur positive Zahlen)

d_0,d_1d_2	$e=-2$	$e=-1$	$e=0$	$e=1$	$e=2$
$\mathbf{1.00_2}$	0.25	0.5	1	2	4
$\mathbf{1.01_2}$	0.3125	0.625	1.25	2.5	5
$\mathbf{1.10_2}$	0.375	0.75	1.5	3	6
$\mathbf{1.11_2}$	0.4375	0.875	1.75	3.5	7

Normalisierte Darstellung

Beispiel $F^*(2, 3, -2, 2)$ (nur positive Zahlen)

d_0,d_1d_2	$\mathbf{e=-2}$	$e=-1$	$e=0$	$e=1$	$e=2$
1.00_2	0.25	0.5	1	2	4
1.01_2	0.3125	0.625	1.25	2.5	5
1.10_2	0.375	0.75	1.5	3	6
1.11_2	0.4375	0.875	1.75	3.5	7

Normalisierte Darstellung

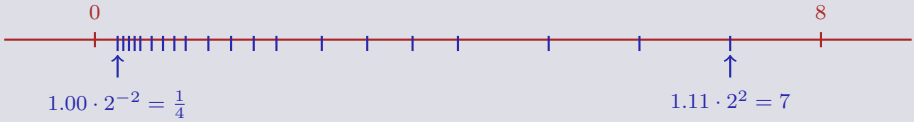
Beispiel $F^*(2, 3, -2, \mathbf{2})$ (nur positive Zahlen)

$d_0{\bullet}d_1d_2$	$e=-2$	$e=-1$	$e=0$	$e=1$	$\mathbf{e=2}$
1.00_2	0.25	0.5	1	2	4
1.01_2	0.3125	0.625	1.25	2.5	5
1.10_2	0.375	0.75	1.5	3	6
1.11_2	0.4375	0.875	1.75	3.5	7

Normalisierte Darstellung

Beispiel $F^*(2, 3, -2, 2)$ (nur positive Zahlen)

$d_0 \bullet d_1 d_2$	$e = -2$	$e = -1$	$e = 0$	$e = 1$	$e = 2$
1.00_2	0.25	0.5	1	2	4
1.01_2	0.3125	0.625	1.25	2.5	5
1.10_2	0.375	0.75	1.5	3	6
1.11_2	0.4375	0.875	1.75	3.5	7



Binäre und dezimale Systeme

- Intern rechnet der Computer mit $\beta = 2$ (binäres System)
- Literale und Eingaben haben $\beta = 10$ (dezimales System)

Binäre und dezimale Systeme

- Intern rechnet der Computer mit $\beta = 2$ (binäres System)
- Literale und Eingaben haben $\beta = 10$ (dezimales System)

Umrechnung $(0 < x < 2)$

Berechnung der *Binärdarstellung*:

$$x = \sum_{i=-\infty}^{0} b_i 2^i$$

Umrechnung ($0 < x < 2$)

Berechnung der *Binärdarstellung*:

$$x = b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots$$

Umrechnung $(0 < x < 2)$

Berechnung der *Binärdarstellung*:

$$\begin{aligned} x &= b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &= b_0 + 0 \bullet b_{-1} b_{-2} b_{-3} \ldots \end{aligned}$$

Umrechnung $(0 < x < 2)$

Berechnung der *Binärdarstellung*:

$$\begin{aligned} x &= b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &= b_0 + 0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &\Longrightarrow \end{aligned}$$

Umrechnung $(0 < x < 2)$

Berechnung der *Binärdarstellung*:

$$\begin{aligned} x &= b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &= b_0 + 0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &\Longrightarrow \\ (x - b_0) &= 0 \bullet b_{-1} b_{-2} b_{-3} b_{-4} \ldots \end{aligned}$$

Umrechnung $(0 < x < 2)$

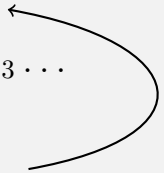
Berechnung der *Binärdarstellung*:

$$\begin{aligned} x &= b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &= b_0 + 0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\ &\Longrightarrow \\ 2 \cdot (x - b_0) &= b_{-1} \bullet b_{-2} b_{-3} b_{-4} \ldots \end{aligned}$$

Umrechnung $(0 < x < 2)$

Berechnung der *Binärdarstellung*:

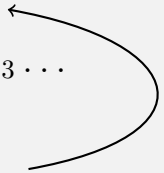
$$
\begin{aligned}
x &= b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\
&= b_0 + 0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\
&\Longrightarrow \\
2 \cdot (x - b_0) &= b_{-1} \bullet b_{-2} b_{-3} b_{-4} \ldots
\end{aligned}
$$



Umrechnung $(0 < x < 2)$

Berechnung der *Binärdarstellung*:

$$
\begin{aligned}
x &= b_0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\
&= b_0 + 0 \bullet b_{-1} b_{-2} b_{-3} \ldots \\
&\Longrightarrow \\
2 \cdot (x - b_0) &= b_{-1} \bullet b_{-2} b_{-3} b_{-4} \ldots
\end{aligned}
$$



Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$		

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$		

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$		

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$		

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6
1.6	$b_{-4} = \mathbf{1}$		

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6
1.6	$b_{-4} = \mathbf{1}$	0.6	1.2

Binärdarstellung von 1.1

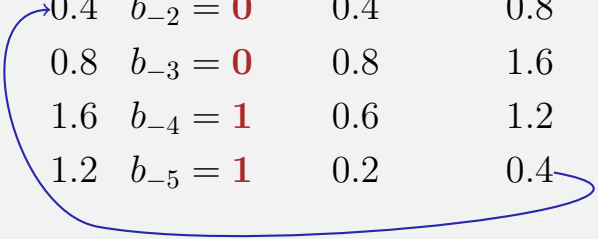
x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6
1.6	$b_{-4} = \mathbf{1}$	0.6	1.2
1.2	$b_{-5} = \mathbf{1}$		

Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6
1.6	$b_{-4} = \mathbf{1}$	0.6	1.2
1.2	$b_{-5} = \mathbf{1}$	0.2	0.4

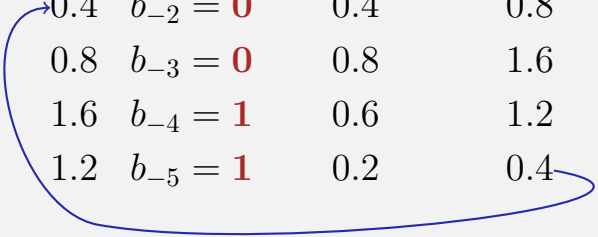
Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6
1.6	$b_{-4} = \mathbf{1}$	0.6	1.2
1.2	$b_{-5} = \mathbf{1}$	0.2	0.4



Binärdarstellung von 1.1

x	b_i	$x - b_i$	$2(x - b_i)$
1.1	$b_0 = \mathbf{1}$	0.1	0.2
0.2	$b_{-1} = \mathbf{0}$	0.2	0.4
0.4	$b_{-2} = \mathbf{0}$	0.4	0.8
0.8	$b_{-3} = \mathbf{0}$	0.8	1.6
1.6	$b_{-4} = \mathbf{1}$	0.6	1.2
1.2	$b_{-5} = \mathbf{1}$	0.2	0.4



$\Rightarrow 1.0\overline{0011}$, periodisch, *nicht* endlich

Binärdarstellungen von 1.1 und 0.1

- sind nicht endlich $\Rightarrow$ Fehler bei der Konversion
- `1.1f` und `0.1f`: *Approximationen* von 1.1 und 0.1

$$\begin{aligned} \texttt{1.1} &= \underline{1.1000000000000000}888178\ldots \\ \texttt{1.1f} &= \underline{1.1000000}238418\ldots \end{aligned}$$

Binärdarstellungen von 1.1 und 0.1

- sind nicht endlich $\Rightarrow$ Fehler bei der Konversion
- `1.1f` und `0.1f`: *Approximationen* von 1.1 und 0.1

$$\begin{aligned} \texttt{1.1} &= 1.1000000000000000888178\ldots \\ \texttt{1.1f} &= 1.1000000238418\ldots \end{aligned}$$

Binärdarstellungen von 1.1 und 0.1

- sind nicht endlich $\Rightarrow$ Fehler bei der Konversion
- `1.1f` und `0.1f`: *Approximationen* von 1.1 und 0.1

$$\begin{aligned} \texttt{1.1} &= \underline{1.1000000000000000}888178\ldots \\ \texttt{1.1f} &= \underline{1.1000000}238418\ldots \end{aligned}$$

Rechnen mit Fliesskommazahlen

ist fast so einfach wie mit ganzen Zahlen.

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{aligned} & 1.111 \cdot 2^{-2} \\ + \quad & 1.011 \cdot 2^{-1} \end{aligned}$$

1. Exponenten anpassen durch Denormalisieren einer Zahl

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{aligned} & 1.111 \cdot 2^{-2} \\ + \quad & 10.110 \cdot 2^{-2} \checkmark \end{aligned}$$

1. Exponenten anpassen durch Denormalisieren einer Zahl

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{array}{rr} & 1.111 \cdot 2^{-2} \\ + & 10.110 \cdot 2^{-2} \\ \hline & \end{array}$$

2. Binäre Addition der Signifikanden

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{aligned} & 1.111 \cdot 2^{-2} \\ + \quad & 10.110 \cdot 2^{-2} \\ \hline = \; & 100.101 \cdot 2^{-2} \checkmark \end{aligned}$$

2. Binäre Addition der Signifikanden

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{aligned} & 1.111 \cdot 2^{-2} \\ + \quad & 10.110 \cdot 2^{-2} \\ \hline & = 100.101 \cdot 2^{-2} \end{aligned}$$

3. Renormalisierung

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{aligned} & 1.111 \cdot 2^{-2} \\ + \quad & 10.110 \cdot 2^{-2} \\ \hline & = 1.00101 \cdot 2^{0} \checkmark \end{aligned}$$

3. Renormalisierung

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{array}{rr} & 1.111 \cdot 2^{-2} \\ + & 10.110 \cdot 2^{-2} \\ \hline & = 1.00101 \cdot 2^{0} \end{array}$$

4. Runden auf p signifikante Stellen, falls nötig

Rechnen mit Fliesskommazahlen

Beispiel ($\beta = 2$, $p = 4$):

$$\begin{aligned} & 1.111 \cdot 2^{-2} \\ + \quad & 10.110 \cdot 2^{-2} \\ \hline & = 1.001 \cdot 2^{0} \checkmark \end{aligned}$$

4. Runden auf p signifikante Stellen, falls nötig

Der IEEE Standard 754

- legt Fliesskommazahlensysteme und deren Rundungsverhalten fest
- wird fast überall benutzt

Der IEEE Standard 754

- legt Fliesskommazahlensysteme und deren Rundungsverhalten fest
- wird fast überall benutzt

Der IEEE Standard 754

- Single precision (`float`) Zahlen:

  $F^*(2, 24, -126, 127)$ plus $0, \infty, \ldots$

- Double precision (`double`) Zahlen:

  $F^*(2, 53, -1022, 1023)$ plus $0, \infty, \ldots$

- Alle arithmetischen Operationen runden das *exakte* Ergebnis auf die nächste darstellbare Zahl

Der IEEE Standard 754

- Single precision (`float`) Zahlen:

  $F^*(2, 24, -126, 127)$ plus $0, \infty, \ldots$

- Double precision (`double`) Zahlen:

  $F^*(2, 53, -1022, 1023)$ plus $0, \infty, \ldots$

- Alle arithmetischen Operationen runden das *exakte* Ergebnis auf die nächste darstellbare Zahl

32-bit Darstellung einer Fliesskommazahl

$\pm$ Exponent Mantisse

$$\pm \begin{matrix} 2^{-126}, \ldots, 2^{127} \\ 0, \infty, \ldots \end{matrix} \qquad \begin{matrix} 1.00000000000000000000000 \\ \vdots \\ 1.11111111111111111111111 \end{matrix}$$

Regel 1

Teste keine gerundeten Fliesskommazahlen auf Gleichheit!

```
for (float i = 0.1; i != 1.0; i += 0.1){
    Out.println(i);
}
```

Fliesskomma-Richtlinien

Regel 1

Teste keine gerundeten Fliesskommazahlen auf Gleichheit!

```
for (float i = 0.1; i != 1.0; i += 0.1){
    Out.println(i);
}
```

Regel 1

Teste keine gerundeten Fliesskommazahlen auf Gleichheit!

```
for (float i = 0.1; i != 1.0; i += 0.1){
    Out.println(i);
}
```

Endlosschleife, weil i niemals exakt 1 ist!

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

$$\begin{aligned} & 1.000 \cdot 2^5 \\ +& 1.000 \cdot 2^0 \end{aligned}$$

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

$$
\begin{aligned}
& 1.000 \cdot 2^5 \\
+ & 1.000 \cdot 2^0 \\
= & \ 1.00001 \cdot 2^5
\end{aligned}
$$

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

$$
\begin{aligned}
& 1.000 \cdot 2^5 \\
+ & 1.000 \cdot 2^0 \\
= & \ 1.00001 \cdot 2^5 \\
\text{“=”} & \ 1.000 \cdot 2^5 \quad \text{(Rundung auf 4 Stellen)}
\end{aligned}
$$

Regel 2

Addiere keine zwei Zahlen sehr unterschiedlicher Grösse!

$$
\begin{aligned}
& 1.000 \cdot 2^5 \\
+ & 1.000 \cdot 2^0 \\
= & \ 1.00001 \cdot 2^5 \\
\text{“=”} & \ 1.000 \cdot 2^5 \quad \text{(Rundung auf 4 Stellen)}
\end{aligned}
$$

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.println("even");
} else {
    Out.println("odd");
}
```

Wo wollen wir hin?

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

Verhalten hängt ab vom Wert eines *Booleschen Ausdrucks*

Wo wollen wir hin?

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

Verhalten hängt ab vom Wert eines *Booleschen Ausdrucks*

Boolesche Werte in der Mathematik

Boolesche Ausdrücke können zwei mögliche Werte annehmen:

$$F \text{ oder } T$$

Boolesche Werte in der Mathematik

Boolesche Ausdrücke können zwei mögliche Werte annehmen:

$$F \text{ oder } T$$

- F entspricht *„falsch”*
- T entspricht *„wahr”*

Der Typ `boolean` in Java

- Repräsentiert *Wahrheitswerte*

Der Typ `boolean` in Java

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

Der Typ `boolean` in Java

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

```
boolean b = true; // Variable mit Wert true (wahr)
```

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

$$f : \{F, T\}^2 \to \{F, T\}$$

- F entspricht „falsch”.
- T entspricht „wahr”.

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

- “Logisches Und”

$$f : \{F, T\}^2 \to \{F, T\}$$

- F entspricht „falsch”.
- T entspricht „wahr”.

x	y	$\mathrm{AND}(x, y)$
F	F	F
F	T	F
T	F	F
T	T	T

Logischer Operator &&

`a && b` (logisches Und)

$$\texttt{boolean} \times \texttt{boolean} \to \texttt{boolean}$$

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”

$$f : \{F, T\}^2 \to \{F, T\}$$

- F entspricht „falsch”.
- T entspricht „wahr”.

x	y	$\mathrm{OR}(x, y)$
F	F	F
F	T	T
T	F	T
T	T	T

Logischer Operator `||`

`a || b` (logisches Oder)

$$\texttt{boolean} \times \texttt{boolean} \to \texttt{boolean}$$

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”

$$f : \{F, T\} \to \{F, T\}$$

x	$\mathrm{NOT}(x)$
F	T
T	F

- F entspricht „falsch”.
- T entspricht „wahr”.

Logischer Operator !

`!b` (logisches Nicht)

`boolean` $\rightarrow$ `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)

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)
```

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)
```

x oder y, und nicht beide

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)
```

x oder y, und nicht beide

```
(x || y)      && (!x || !y)
```

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)    x oder y, und nicht beide

(x || y)      && (!x || !y)   x oder y, und eines nicht
```

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)
```
x oder y, und nicht beide

```
(x || y)      && (!x || !y)
```
x oder y, und eines nicht

```
!(!x && !y)  && !(x && y)
```

Anwendung: Entweder ... Oder (XOR)

```
(x || y)       && !(x && y)    x oder y, und nicht beide

(x || y)       && (!x || !y)   x oder y, und eines nicht

!(!x && !y)    && !(x && y)    nicht keines, und nicht beide
```

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)    x oder y, und nicht beide

(x || y)      && (!x || !y)   x oder y, und eines nicht

!(!x && !y)   && !(x && y)    nicht keines, und nicht beide

!(!x && !y || x && y)
```

Anwendung: Entweder ... Oder (XOR)

```
(x || y)      && !(x && y)     x oder y, und nicht beide

(x || y)      && (!x || !y)    x oder y, und eines nicht

!(!x && !y)   && !(x && y)     nicht keines, und nicht beide

!(!x && !y || x && y)          nicht: keines oder beide
```

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