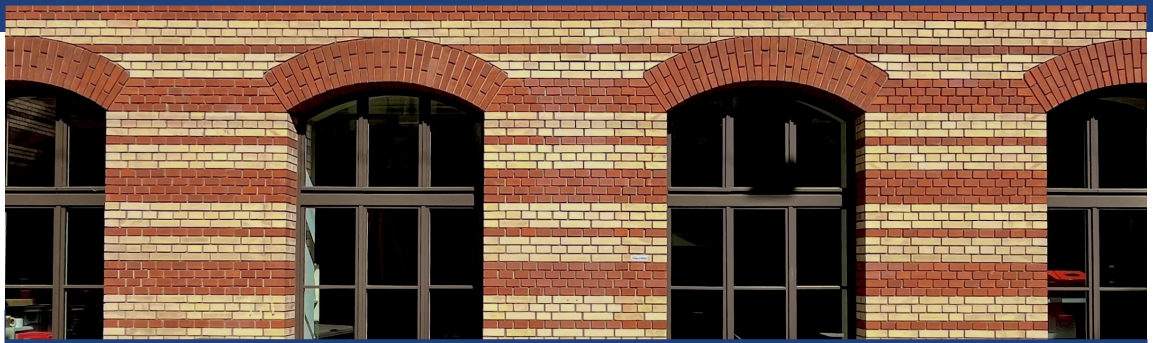


Informatik

Binärdarstellung, Fliesskommazahlen

Adel Gavranović

ETH Zürich — 2026

Überblick

`const int& foo(int& i)`
Klammern bei `if`/`else`
Rekursion: ein zweiter Versuch
Von Dezimal zu Binär
Binärdarstellung von Kommazahlen
Normalisierte Fliesskommazahlensysteme
Extra: Negative ganze Zahlen
Extra: Ein 10-Bit-Fliesskommatyp

adelgavranovic.ch

Webpage

Mail

1. Follow-up

Follow-up aus letzter Übungsstunde

1. `const int& foo(int& i)` (Woche 6, Aufgabe 1)
2. Klammern bei `if`/`else`
3. Rekursion: ein zweiter Versuch
4. Eure Meinung zu den Übungsstunden (am Schluss)

2. `const int& foo(int& i)`

Woche 6, Aufgabe 1

```
const int& foo(int& i) {
  return ++i;
}
```

a) Kompilierfehler? Nein. Eine `const`-Referenz zurückzugeben ist erlaubt.
b) Wohldefiniert? Ja. Die Rückgabe ist ein Alias für die Variable hinter `i`, und die gibt es nach dem Aufruf noch.
c) Nachbedingung? `POST:` erhöht `i` um 1 und gibt eine `const`-Referenz auf `i` zurück.

Ausprobiert

```cpp
int const& foo(int& i) {
  std::cout << " -- foo called -- " << "\n";
  return ++i;
}

int main() {
  int test = 42;
  std::cout << "test       = " << test << "\n";
  int secondtest = foo(test);
  std::cout << "test       = " << test << "\n";
  std::cout << "secondtest = " << secondtest << "\n";

  const int& reftotest = foo(test);
  std::cout << "reftotest  = " << reftotest << "\n";
  ++test;
  std::cout << "reftotest  = " << reftotest << "\n";
  std::cout << "secondtest = " << secondtest << "\n";
  return 0;
}
```

Ausgabe:

```
test       = 42
 -- foo called --
test       = 43
secondtest = 43
 -- foo called --
reftotest  = 44
reftotest  = 45
secondtest = 43
```

`secondtest` ist eine Kopie, `reftotest` ein Alias für `test`.

`int const& = const int&`

Welcher Typ darf das Resultat entgegennehmen?

```
int         a = foo(test);  // copy
const int   b = foo(test);  // copy, read-only
const int&  c = foo(test);  // alias for test, read-only
int&        d = foo(test);  // error
```

- `a`, `b`: Kopien. Was mit der Kopie passiert, betrifft `test` nicht
- `c`: Alias, aber nur zum Lesen, wie versprochen
- `d`: über `d` könnte man `test` ändern. `foo` hat aber “nur lesen” versprochen, also verbietet es der Compiler

```
error:  binding reference of type ’int&’ to ’const int’ discards
qualifiers
```

Fragen/Unklarheiten?

3. Klammern bei `if`/`else`

Klammern bei `if`/`else`

```
if (leap) {
    return 366;
  }
    else {
      return 365;
      }
```



```
if (leap) {
    return 366;
}
else {
    return 365;
}
```



```
if (leap) {
    return 366;
} else {
    return 365;
}
```



- Der Körper ist genau eine Stufe eingerückt, die schliessende Klammer steht unter dem **`if`**
- `} else {` gehört auf eine Zeile
- Aber wie immer gilt: Standardisiert ist besser als optimal.

Style Guide

4. Rekursion: ein zweiter Versuch

Rezept für rekursive Funktionen

1. **Basisfall**: den kleinsten Fall direkt lösen
2. **Rekursionsschritt**: Problem verkleinern, sich selbst aufrufen, aus dem Resultat die Lösung bauen
3. **Vertrauen**: annehmen, dass der kleinere Aufruf schon stimmt
4. **Terminierung**: jeder Aufruf kommt dem Basisfall näher

Nochmals probieren

Öffnet “The Towers of Hanoi” auf **code** expert und löst sie nochmals, diesmal mit dem Rezept.

5 Minuten, alleine oder zu zweit. Danach schauen wir die Lösung zusammen an.

Fragen/Unklarheiten?

5. Feedback zu **code** expert

Allgemeines bezüglich **code** expert

- Woche 8, “Reverse Digits”: nur rekursiv. `for` und `while` sind gesperrt, ohne Rekursion gibt es 0 XP
- Challenge Exercise 1: Abgabe bis nächsten Montag, 9. November
- Bei einer Aufgabe steckengeblieben? Kommt ins Study Center (jeden Mittwoch, Infos auf der Study-Center-Webseite)! Study Center

Fragen zu den Übungen der letzten Woche?

- Ich kann es nur wärmstens empfehlen, Fragen im Code zu stellen
- Habe ich jemals welche übersehen?

6. Lernziele

Ziele

- ☐ Eine nicht-ganzzahlige Dezimalzahl in Binärdarstellung umwandeln können

7. Zusammenfassung

Getting on the same page

- Was wurde diese Woche behandelt?
- (Diese Woche gibt es keine Zusammenfassung)

8. Von Dezimal zu Binär

Ein Algorithmus

Wie wandelt man eine Dezimalzahl in Binärdarstellung um? Zum Beispiel 61?

Durch 2 teilen, den Rest notieren. Mit dem Quotienten weitermachen, bis 0 übrig bleibt.

$$
\begin{aligned}
61 &= 2 \cdot 30 &&+ 1 \\
30 &= 2 \cdot 15 &&+ 0 \\
15 &= 2 \cdot 7 &&+ 1 \\
7 &= 2 \cdot 3 &&+ 1 \\
3 &= 2 \cdot 1 &&+ 1 \\
1 &= 2 \cdot 0 &&+ 1
\end{aligned}
$$

Von unten nach oben gelesen: $61_{10} = 111101_2$

Rekursiv!

- Für 61 machen wir genau dasselbe nochmals, mit $30 = 61/2$: Rekursion
- Zurück zu dezimal: die Stellenwerte der Einsen addieren

$$\begin{array}{lcccccc} \text{bin} & 1 & 1 & 1 & 1 & 0 & 1 \\ \text{dec} & 32 & 16 & 8 & 4 & 0 & 1 \quad = 61 \end{array}$$

- So speichern `int` und `unsigned int` ihre Werte, mit fixer Anzahl Bits. Mit 4 Bits:
  - `unsigned int`: höchstens $1111_2 = 15$
  - `int`: ein Bit fürs Vorzeichen, also höchstens $0111_2 = 7$

Die Vorlesung rät von `unsigned int` ab: gemischte Ausdrücke mit `int` führen oft zu Fehlern.

Aufgabe “From decimal to binary representation”

- Öffnet “Code Examples” auf **code** expert
- Geht zu “Week 8: Exercise Session” und öffnet “From decimal to binary representation”
- Implementiert den Algorithmus von vorhin

Nur nicht-negative ganze Zahlen.

Zu zweit, 10 Minuten.

Mögliche Lösung

```cpp
#include <iostream>

void printBinary(int number) {
  if (number >= 2) {
    printBinary(number / 2);
  }
  std::cout << (number % 2);
}

int main() {
  int number;
  std::cout << "Enter a non-negative integer number:\n";
  std::cin >> number;

  printBinary(number);

  return 0;
}
```

- Basisfall: einzelne Ziffer (0 oder 1)
- Erst die höheren Bits ausgeben (Aufruf), dann den eigenen Rest
- So kommt “von unten nach oben” gratis, wie bei “Partial Sum” letzte Woche

Fragen/Unklarheiten?

9. Binärdarstellung von Kommazahlen

Stellen nach dem Punkt

bin	1	1	1	1	.	1	1	1
dec	8	4	2	1	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

Beispiel

$$101.011_2 = 4 + 1 + \tfrac{1}{4} + \tfrac{1}{8} = 5.375_{10}$$

Dezimal zu binär

Vorgehen

1. b_i: Ziffer vor dem Punkt
2. $x - b_i$
3. mal 2, nächste Zeile

Die b_i von oben nach unten sind die Binärziffern.

x	b_i	$x - b_i$	$2 \cdot (x - b_i)$
1.9	1	0.9	1.8
1.8	1	0.8	1.6
$\mathbf{1.6}$	1	0.6	1.2
1.2	1	0.2	0.4
0.4	0	0.4	0.8
0.8	0	0.8	1.6
$\mathbf{1.6}$	1	0.6	1.2

$\mathbf{1.6}$ wiederholt sich: $1.9_{10} = 1.1\overline{1100}_2$

Frage

Warum hat 0.1 dezimal eine endliche Darstellung, binär aber nicht?

$0.1 = \frac{1}{10}$ und $10 = 2 \cdot 5$. Endlich darstellbar sind nur Brüche, deren Nenner aus den Primfaktoren der Basis besteht. Die Basis 10 hat 2 und 5, die Basis 2 nur die 2.

Aufgabe “Binary Representation”

Aufgabe

Berechnet die Binärdarstellung von

1. 0.25
2. 11.1

Alleine, 10 Minuten.

Lösung

x	b_i	$x - b_i$	$2 \cdot (x - b_i)$
0.25	0	0.25	0.5
0.5	0	0.5	1
1	1	0	0

$0.25_{10} = 0.01_2$

$11.1 = 11 + 0.1$, $11_{10} = 1011_2$

x	b_i	$x - b_i$	$2 \cdot (x - b_i)$
0.1	0	0.1	0.2
0.2	0	0.2	0.4
0.4	0	0.4	0.8
0.8	0	0.8	1.6
1.6	1	0.6	1.2
1.2	1	0.2	0.4
0.4	0	0.4	0.8

$11.1_{10} = 1011.0\overline{00011}_2$

Fragen/Unklarheiten?

10. Normalisierte Fliesskommazahlensysteme

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

Welche Zahlen beschreibt $F^*(\beta, p, e_{\min}, e_{\max})$?

$$\pm\, d_0.d_1 d_2 \ldots d_{p-1} \cdot \beta^e$$

- $\beta \geq 2$: Basis
- $p \geq 1$: Anzahl Ziffern
- $e \in \{e_{\min}, \ldots, e_{\max}\}$
- $d_i \in \{0, \ldots, \beta - 1\}$
- $d_0 \neq 0$: normalisiert (*)

Warum $d_0 \neq 0$? Wie in der wissenschaftlichen Notation: $2.0 \cdot 10^3 = 0.2 \cdot 10^4$.
Mit genau einer Ziffer $\neq 0$ vor dem Punkt ist die Darstellung eindeutig.

Liegen diese Zahlen in $F^*(2, 4, -2, 2)$?

Zahl	dezimal	
$0.000 \cdot 2^1$	0	✗ nicht normalisierbar
$1.000 \cdot 2^1$	2	✓
$1.001 \cdot 2^{-1}$	0.5625	✓
$1.0001 \cdot 2^{-1}$	0.53125	✗ 5 Ziffern, aber $p = 4$
$1.111 \cdot 2^{-2}$	0.46875	✓
$1.111 \cdot 2^5$	60	✗ $5 \notin \{-2, \ldots, 2\}$

Fragen/Unklarheiten?

Aufgabe “Some numbers in $F^*(2, 4, -2, 2)$”

Aufgabe

Gebt die folgenden Zahlen aus $F^*(2, 4, -2, 2)$ an:

1. die grösste
2. die kleinste
3. die kleinste nicht-negative

Wie viele Zahlen enthält $F^*(2, 4, -2, 2)$?

Alleine, 10 Minuten. Danach diktiert ihr mir die Lösung.

Lösung

1. grösste: $1.111 \cdot 2^2 = 7.5$
2. kleinste: $-1.111 \cdot 2^2 = -7.5$
3. kleinste nicht-negative: $1.000 \cdot 2^{-2} = 0.25$

Anzahl: $\underbrace{2}_{\pm} \cdot \underbrace{2^3}_{d_1 d_2 d_3} \cdot \underbrace{5}_{e} = 80$

Keine Zahl wird doppelt gezählt: die normalisierte Darstellung ist eindeutig.

Fragen/Unklarheiten?

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

1. Beide auf denselben Exponenten bringen
2. Binär addieren
3. Renormalisieren
4. Runden, falls nötig

Beispiel in $F^*(2, 6, -2, 3)$

$$
\begin{aligned}
& 1.125 + 9.25 \\
&= 1.001_2 + 1001.01_2 && \text{schon gleicher Exponent } (2^0) \\
&= 1010.011_2 && \text{addieren} \\
&= 1.010011 \cdot 2^3 && \text{renormalisieren: 7 Ziffern, aber } p = 6 \\
&\approx 1.01010 \cdot 2^3 && \text{runden} \\
&= 1010.1_2 = 10.5 && \neq 10.375
\end{aligned}
$$

Warum 10.5 und nicht 10.375?

- 10.375 liegt nicht in $F^*(2, 6, -2, 3)$. Die Nachbarn sind 10.25 und 10.5
- 10.375 liegt genau dazwischen: hier wird aufgerundet
- Deshalb: gerundete Fliesskommazahlen nie auf Gleichheit testen

Aufgabe “Computations in $F^*(2, 4, -2, 2)$”

Aufgabe

Addiert $1.001 \cdot 2^{-1}$ $(= 0.5625)$ und $1.111 \cdot 2^{-2}$ $(= 0.46875)$ in $F^*(2, 4, -2, 2)$.

Alleine, 5 Minuten.

Lösung

1. Exponent -1: $1.001 \cdot 2^{-1}$ und $0.1111 \cdot 2^{-1}$
2. Addieren:
$$\begin{array}{rr} & 1.001 \cdot 2^{-1} \\ + & 0.1111 \cdot 2^{-1} \\ \hline & 10.0001 \cdot 2^{-1} \end{array}$$
3. Renormalisieren: $1.00001 \cdot 2^{0}$
4. Runden: $1.000 \cdot 2^{0} = 1$

Exakt wäre 1.03125: die Präzision $p = 4$ reicht dafür nicht.

Fragen/Unklarheiten?

11. Extra: Negative ganze Zahlen

Wie speichert man -1?

Wir wollen $x + (-x) = 0$. Mit 4 Bits:

$$\begin{array}{rr} & 0001 \\ + & ???? \\ \hline & 0000 \end{array}$$

$$\begin{array}{rr} & 0001 \\ + & 1111 \\ \hline & 10000 \end{array}$$

Das fünfte Bit hat keinen Platz und fällt weg.

Zweierkomplement

Und für ein beliebiges negatives x?

1. $|x|$ binär
2. Bits kippen
3. $+1$

-2: $0010 \to 1101 \to 1110$

4 Bits: $-8 = 1000$ bis $7 = 0111$

bin	**int**	bin	**int**
0000	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
0011	3	1011	-5
0100	4	1100	-4
0101	5	1101	-3
0110	6	1110	-2
0111	7	1111	-1

Fragen/Unklarheiten?

12. Extra: Ein 10-Bit-Fliesskommatyp

Erster Versuch

Wie die wissenschaftliche Notation, nur binär:

$$d_0.d_1d_2d_3d_4d_5 \cdot 2^e$$

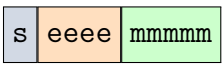
- 1 Bit vor dem Punkt, 5 danach, 4 für den Exponenten
- Exponent: 0000 bis 1111, minus *Bias* $8 \Rightarrow e \in \{-8, \ldots, 7\}$

$\Rightarrow F(2, 6, -8, 7)$

Den Bias macht IEEE 754 genauso.

Vorzeichen und verstecktes Bit

- Normalisiert ist d_0 immer 1: wir müssen es nicht speichern (*hidden bit*)
- Das freie Bit wird zum Vorzeichen



$\Rightarrow F^*(2, 6, -8, 7)$

grösste:	$+1.11111 \cdot 2^7 = 252$
kleinste:	$-1.11111 \cdot 2^7 = -252$
kleinste positive:	$+1.00000 \cdot 2^{-8} = \frac{1}{256}$

Und die Null?

Mit dem versteckten Bit gibt es keine 0 mehr. Also reservieren wir den Exponenten -8 (0000) für Spezialwerte:

0	0000	00000	0
0	0000	00001	$+\infty$ (mit $s = 1$: $-\infty$)
0	0000	00011	NaN

$\Rightarrow F^*(2, 6, -7, 7)$

`float` (32 Bit) und `double` (64 Bit) funktionieren genauso.

IEEE-754 Converter

Fragen/Unklarheiten?

13. Outro

Eure Meinung

- Sollen wir öfter Code aus den letzten Wochen zusammen anschauen?
- Was fehlt euch in den Übungsstunden?
- Wovon machen wir zu viel?

Allgemeine Fragen?

Bis nächste Woche!