



Übungsstunde — Informatik — 04

Adel Gavranović

Fliesskommazahlen: Binärrepräsentation, Guidelines, Vergleiche;
Normalisierte Fliesskommazahlensysteme

Übersicht

Intro

Repetition

Binärdarstellung

Normalisiertes Fließkommazahlensystem

Floating Point Guidelines

Fließkommazahlen vergleichen

Alte Prüfungsfrage



`n.ethz.ch/~agavranovic`

 Material

 Webpage

 Mail

1. Intro

Intro

- Bin seit Sonntag krank und in Isolation
- Hoffe euch hat die Session mit Ioana¹ gefallen! :)

¹  Ioana's Webpage

2. Repetition

Expressions

Aufgabe

Evaluierere die folgenden Expressions:

1. `5 < 4 < 1`

2. `true > false`

Lösung 1

`5 < 4 < 1`

`(5 < 4) < 1`

`false < 1`

`0 < 1`

`true`

Lösung 2

`true > false`

`1 > 0`

`true`

Binärdarstellung ...aber welche?

Math. Dec	Math. Bin	int	unsigned int
42_{10}	101010_2	0101010	101010
-42_{10}	-101010_2	1010110	-

`int` speichert Zahlen in der *two's complement Representation*, daher die führende 0.

Binäre Arithmetik

Aufgabe

1. Konvertiert die Ganzzahlen $a = 4$ und $b = 7$ in ihre Binärdarstellung (*nicht* Two's Complement)
2. Addiert die zwei in ihren Binärdarstellungen
3. Konvertiert das Resultat zurück in Dezimaldarstellung

Lösung

$$a = 4_{10} = 100_2$$

$$b = 7_{10} = 111_2$$

$$100_2 + 111_2 = 1011_2$$

$$1011_2 = 11_{10}$$

Fragen/Unklarheiten?

3. Binärdarstellung

Binärdarstellung

binary	1	1	1	1	.	1	1	1
decimal	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
	8	4	2	1	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

Beispiel

$$101.011_2 = 2^2 + 2^0 + 2^{-2} + 2^{-3} = 4 + 1 + \frac{1}{4} + \frac{1}{8} = 5.375_{10}$$

Umwandlung in Binärdarstellung

Vorgehen (Aus der Vorlesung)

1. Zahl aufschreiben
2. Erste Ziffer der Zahl aufschreiben
3. Berechne: Zahl – erste Ziffer
4. Mit 2 multiplizieren und in eine neue Zeile schreiben

Beispiel

$$x = 1.9$$

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
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
1.6	1	0.6	1.2

Die **1.6** bemerkt?

$$1.6_{10} = 1.1\overline{1100}_2 =$$

$$1.11100110011001100110011001100110011$$

Dezimal zu Binär

Aufgabe Berechne die binären Expansionen von 0.25_{10} und 11.1_{10} .

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

$$0.25_{10} = 0.01_2$$

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

$$11.1_{10} = 1011.000\overline{11}_2$$

Fragen/Unklarheiten?

4. Normalisiertes Fließkommazahlensystem

Normalisiertes Fließkommazahlensystem

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

...beschreibt Zahlen der Form:

- * Normalisiert ($d_0 \neq 0$)
- $\beta \geq 2$ Basis
- $p \geq 1$ Präzision (Anzahl Ziffern)
- e_{\min} kleinstmöglicher Exponent
- e_{\max} grösstmöglicher Exponent

$$\pm d_0.d_1d_2d_3 \dots d_{p-1} \cdot \beta^e$$

wobei:

$$d_i \in \{0, \dots, b - 1\}$$

$$d_0 \neq 0$$

$$e \in [e_{\min}, e_{\max}]$$

Übungen

Übungen

Sind die folgenden Zahlen im Set $F^*(2, 4, -2, 2)$?

$$0.000_2 \cdot 2^1 = 0_{10}$$

$$1.000_2 \cdot 2^1 = 2_{10}$$

$$1.001_2 \cdot 2^{-1} = 0.5625_{10}$$

$$1.0001_2 \cdot 2^{-1} = 0.53125_{10}$$

$$1.111_2 \cdot 2^{-2} = 0.46875_{10}$$

$$1.111_2 \cdot 2^5 = 60_{10}$$

Lösungen

*in F^**

$$1.000_2 \cdot 2^1 = 2_{10}$$

$$1.001_2 \cdot 2^{-1} = 0.5625_{10}$$

$$1.111_2 \cdot 2^{-2} = 0.46875_{10}$$

*nicht in F^**

$$0.000_2 \cdot 2^1 \text{ nicht "normalisierbar"}$$

$$1.0001_2 \cdot 2^{-1} \quad 5 > p = 4$$

$$1.111_2 \cdot 2^5 \quad 5 \notin [-2, 2]$$

mehr Übungen

Aufgabe

Nenne die folgenden Zahlen in $F^*(2, 4, -2, 2)$ als Dezimalzahlen

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

Lösung

grösste: $1.111 \cdot 2^2 = 7.5_{10}$
kleinste: $-1.111 \cdot 2^2 = -7.5_{10}$
kleinste > 0 : $1.000 \cdot 2^{-2} = 0.25_{10}$

Trick

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

Grösste: $1.11 \dots 1 \cdot 2^{e_{\max}}$
Kleinste: $-\text{Grösste}$
Kleinste > 0 : $1.00 \dots 0 \cdot 2^{e_{\min}}$

Normalisiertes Fließkommazahlensystem

Frage

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

Antwort

Für einen fixierten Exponenten gibt es immer drei Ziffern, die man frei variieren kann und für alle Zahlen gibt es auch eine jeweils negative in der Menge F^* . Das resultiert in $2 \cdot 2^3 = 16$ Zahlen pro Exponenten. Es gibt 5 mögliche Exponenten, daher also $5 \cdot 16 = 80$ Zahlen. Merke, dass keine Zahl doppelt gezählt wird, da jede NFP eindeutig (*unique*) ist.

Fragen/Unklarheiten?

Arithmetik in F^*

Floats addieren

1. Beide zum gleichen Exponenten umformen
2. In Binärdarstellung addieren
3. Summe renormalisieren
4. Runden, falls nötig

Beispiel

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

$$1.125_{10} + 9.25_{10}$$

$$1.001_2 + 1001.01_2$$

$$1010.011$$

$$1.010011 \cdot 2^3$$

$$1.01010 \cdot 2^3$$

$$1.01010_2 \cdot 2^3 = 1010.10_2 = 10.5_{10}$$

(bereits gleicher Exponent ($\cdot 2^0$))

Addition

Renormalisierung, e und p anpassen

Runden: 1 rauf (und ggf. übertragen), 0 runter

$\neq 10.375_{10}$

Wieso 10.5 und nicht 10.375?

Einfach, weil die exakte Zahl 10.375 nicht im gegebenen F^* dargestellt werden *kann*. Die nächste Zahl, die in F^* ist, ist 10.5. Das ist der Grund, wieso Floats gefährlich sein können. Deshalb müssen wir auch die *Floating Point Guidelines* befolgen.

Es ist nicht 10.25, weil wir in diesem Fall aufrunden, obwohl die Differenz bei beiden 10.25 und 10.5 zu 10.375 bei 0.125 liegt.

Übung

Übung

Addiere

$$1.001 \cdot 2^{-1} = 0.5625_{10}$$

zu

$$1.111 \cdot 2^{-2} = 0.46875_{10}$$

in

$$F^*(2, 4, -2, 2).$$

Lösung

1. Bringe beide auf den gleichen Exponenten, sagen wir -1

$$1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$$

2. Addiere sie in der Binärdarstellung:

$$10.0001 \cdot 2^{-1}$$

3. Renormalisiere:

$$1.00001 \cdot 2^0$$

4. Runde: $1.000 \cdot 2^0 = 1_{10} \neq 1.03125_{10}$

Fragen/Unklarheiten?

5. Floating Point Guidelines

Guidelines

Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

Guideline 1 – Example

Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

This is false

Example:

```
float a = 1.05f;  
if (100*a == 105.0f)  
    std::cout << "no output\n";
```

Guideline 1 – Example

Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

This is false

Example:

```
float a = 1.05f;  
if (100*a == 105.0f)  
    std::cout << "no output\n";
```

Problem:

1.05f not
representable

Guideline 1 – Example

Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

This is false

Example:

```
float a = 1.05f;  
if (100*a == 105.0f)  
    std::cout << "no output\n";
```

Problem:

1.05f not
representable

1.05 = $1.0000110011001100110011001... \cdot 2^0$
(rounding) $\rightarrow 1.049999995231... = 1.0000110011001100110 \cdot 2^0$

24bit

Guidelines

Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

Guideline 2:

«**Avoid** the **addition** of numbers of extremely **different sizes!**»

Guideline 2 – Example

Guideline 2:

«**Avoid the addition** of numbers of extremely **different sizes!**»

Example:

```
float a = 67108864.0f + 1.0f;  
  
if (a > 67108864.0f)  
    std::cout << "This is not output ... \n";
```


Guideline 2 – Example

Guideline 2:

«**Avoid the addition** of numbers of extremely **different sizes!**»

Example:

```
float a = 67108864.0f + 1.0f;  
  
if (a > 67108864.0f)  
    std::cout << "This is not output ... \n";
```

Problem:

Significant too
short

Guideline 2 – Example

Guideline 2:

«**Avoid the addition** of numbers of extremely **different sizes!**»

Example:

```
float a = 67108864.0f + 1.0f;  
  
if (a > 67108864.0f)  
    std::cout << "This is not output ... \n";
```

Problem:

Significant too short

$$\begin{array}{r} 67108864 = \overbrace{1.000000000000000000000000}^{24\text{bit}} \cdot 2^{26} \\ +1 = 0.000000000000000000000001 \cdot 2^{26} \\ \hline 67108865 = 1.000000000000000000000001 \cdot 2^{26} \end{array}$$

Guideline 2 – Example

Guideline 2:

«**Avoid** the **addition** of numbers of extremely **different sizes!**»

Problem:

Significant too short

Example:

```
float a = 67108864.0f + 1.0f;

if (a > 67108864.0f)
    std::cout << "This is not output ... \n";
```

$$\begin{aligned} 67108864 &= \overbrace{1.000000000000000000000000}^{24\text{bit}} \cdot 2^{26} \\ +1 &= 0.000000000000000000000001 \cdot 2^{26} \\ \hline 67108865 &= 1.000000000000000000000001 \cdot 2^{26} \\ (\text{rounding}) \rightarrow 67108864 &= 1.000000000000000000000000 \cdot 2^{26} \end{aligned}$$

Guidelines

Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

Guideline 2:

«**Avoid** the **addition** of numbers of extremely **different sizes!**»

Guideline 3:

«**Avoid** the **subtraction** of numbers of **similar sizes!**»

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

Example:

- Consider sequence $x_{n+1} = 6x_n - 1$

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

Example:

- Consider sequence $x_{n+1} = 6x_n - 1$
- Computing some sequences for given x_0 :

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction of numbers of similar sizes!**»

Example:

- Consider sequence $x_{n+1} = 6x_n - 1$
- Computing some sequences for given x_0 :
 - e.g. $x_0 = 1 \rightarrow x_1 = 5, x_2 = 29, x_3 = 173, \dots$

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

Example:

- Consider sequence $x_{n+1} = 6x_n - 1$
- Computing some sequences for given x_0 :
 - e.g. $x_0 = 1 \quad \rightarrow \quad x_1 = 5, \quad x_2 = 29, \quad x_3 = 173, \quad \dots$
 - e.g. $x_0 = 0.2 \quad \rightarrow \quad x_1 = 0.2, \quad x_2 = 0.2, \quad x_3 = 0.2, \quad \dots$

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

Example:

- Consider sequence $x_{n+1} = 6x_n - 1$
- Computing some sequences for given x_0 :
 - e.g. $x_0 = 1 \quad \rightarrow \quad x_1 = 5, \quad x_2 = 29, \quad x_3 = 173, \quad \dots$
 - e.g. $x_0 = 0.2 \quad \rightarrow \quad x_1 = 0.2, \quad x_2 = 0.2, \quad x_3 = 0.2, \quad \dots$

C++ claims

$x_{14} \approx 622.982$

Guideline 3 – Example

Guideline 3:

«**Avoid** the **subtraction** of numbers of **similar sizes!**»

Example:

- What went wrong?

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction of numbers of similar sizes!**»

Example:

- What went wrong?
 - `float` represents 0.2 as 0.20000000298...
 - Thus: $6 \cdot x_0 - 1 \neq 1.2 - 1$

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

Example:

- What went wrong?
 - `float` represents 0.2 as 0.20000000298...
 - Thus: $6 \cdot x_0 - 1 \neq 1.2 - 1$ but rather:
 - $x_1 = 0.20000004768 \dots$
 - $x_2 = 0.20000028610 \dots$
 - $x_3 = 0.20000171661 \dots$
 - \vdots

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

Example:

- What went wrong?
 - `float` represents 0.2 as 0.20000000298...
 - Thus: $6 \cdot x_0 - 1 \neq 1.2 - 1$ but rather:
 $x_1 = 0.20000004768 \dots$
 $x_2 = 0.20000028610 \dots$
 $x_3 = 0.20000171661 \dots$
⋮

Note how error
increases!

Guideline 3 – Example

Guideline 3:

«**Avoid the subtraction** of numbers of **similar sizes!**»

But why do we subtract two similarly sized floating point numbers when we are comparing them to see if they are (roughly) equal?

In these cases, we do not further use the result from the subtraction in any other calculations. Therefore, the error does not add up over time causing issues as seen in the previous example.

Floating Point Numbers Vergleichen

Kurzgesagt: nicht auf *Gleichheit* prüfen
lieber auf "*innerhalb der Toleranz*" prüfen

```
// Example of "equality" check function for doubles
bool equal(double x, double y, double tol){
    double diff = x - y;
    if(diff < 0){
        diff *= -1;
    }
    return (diff < tol);
}
```

6. Fließkommazahlen vergleichen

The Comparison Problem

- Given `fp1` and `fp2` of type `float` or `double`.

- Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

The Comparison Problem

- Given `fp1` and `fp2` of type `float` or `double`.

- Guideline 1:

«Do **not** test two floating point numbers for **equality**, if at least one of them was rounded before.»

- Thus `fp1 == fp2` should be **avoided**.

The Comparison Problem

- How can we **compare** instead?

The Comparison Problem

- How can we **compare** instead?
- First idea:
Allow for **small differences!**

Given: tolerance value $c > 0$.

fp1 "equals" fp2 whenever $|fp1 - fp2| < c$

(Remark: $|...|$ means absolute value. In C++ it's not available using vertical bars.)

The Comparison Problem

Given: tolerance value $c > 0$.

fp1 "equals" fp2 whenever $|fp1 - fp2| < c$

- Examples (c is 0.001):
 - $fp1 = 10.0$ and $fp2 = 12.0$

(Remark: on this slide = is meant in the mathematical sense.)

The Comparison Problem

Given: tolerance value $c > 0$.

`fp1 "equals" fp2` whenever $|fp1 - fp2| < c$

- Examples (c is 0.001):
 - `fp1 = 10.0` and `fp2 = 12.0`
 $|10.0 - 12.0| = 2.0$

(Remark: on this slide = is meant in the mathematical sense.)

The Comparison Problem

Given: tolerance value $c > 0$.

`fp1 "equals" fp2` whenever $|fp1 - fp2| < c$

- Examples (c is 0.001):
 - `fp1 = 10.0` and `fp2 = 12.0`
 $|10.0 - 12.0| = 2.0 > c$
Thus: **not "equal"**

(Remark: on this slide = is meant in the mathematical sense.)

The Comparison Problem

Given: tolerance value $c > 0$.

`fp1 "equals" fp2` whenever $|fp1 - fp2| < c$

- Examples (c is `0.001`):

- `fp1 = 10.0` and `fp2 = 12.0`
 $|10.0 - 12.0| = 2.0 > c$

Thus: **not "equal"**

- `fp1 = 10.0` and `fp2 = 10.000013`

(Remark: on this slide = is meant in the mathematical sense.)

The Comparison Problem

Given: tolerance value $c > 0$.

`fp1 "equals" fp2` whenever $|fp1 - fp2| < c$

- Examples (c is `0.001`):

- `fp1 = 10.0` and `fp2 = 12.0`
 $|10.0 - 12.0| = 2.0 > c$

Thus: **not "equal"**

- `fp1 = 10.0` and `fp2 = 10.000013`
 $|10.0 - 10.000013| = 0.000013$

(Remark: on this slide = is meant in the mathematical sense.)

The Comparison Problem

Given: tolerance value $c > 0$.

`fp1 "equals" fp2` whenever $|fp1 - fp2| < c$

- Examples (c is 0.001):

- `fp1 = 10.0` and `fp2 = 12.0`
 $|10.0 - 12.0| = 2.0 > c$

Thus: **not "equal"**

- `fp1 = 10.0` and `fp2 = 10.000013`
 $|10.0 - 10.000013| = 0.000013 < c$

Thus: **"equal"**

(Remark: on this slide = is meant in the mathematical sense.)

Exercise

Write the following function:

```
// POST: returns true if and only if
//      |x - y| < tol
bool equals (double x, double y, double tol) {
    ...
}
```

Exercise

For example:

```
// POST: returns true if and only if
//      |x - y| < tol
bool equals (double x, double y, double tol) {
    double diff = x - y;
    if (diff < 0)
        diff *= -1; // absolute value
    return diff < tol;
}
```

Remark

- Comparing absolute differences with a tolerance value is a great first idea!
- (But: for example problems when the numbers are large.)

7. Alte Prüfungsfrage

Prüfungsfrage

Geben Sie ein möglichst knappes normalisiertes Fließkommazahlensystem an, mit welchem sich die folgenden dezimalen Werte gerade noch genau darstellen lassen: jede Verkleinerung von p , e_{\max} oder $-e_{\min}$ muss dazu führen, dass mindestens eine Zahl nicht mehr dargestellt werden kann.

Hinweis: p zählt auch die führende Ziffer.

Tipp: Schreiben Sie sich die normalisierte Binärzahldarstellung der Werte auf, wenn sie für Sie nicht offensichtlich ist.

Werte / *Values*: 2.25, $\frac{1}{8}$, 0.5, 16.5, 2^3

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

$\beta = 2$, $p =$, $e_{\min} =$, $e_{\max} =$.

Provide a smallest possible normalized floating point number system that can still represent the following values exactly: any decrease of the numbers p , e_{\max} or $-e_{\min}$ must imply that at least one of the numbers cannot be represented any more.

Hint: p does also count the leading digit.

Tip: Write down the normalized binary representation of the values, if it is not obvious for you.

Prüfungsfrage

Geben Sie ein möglichst knappes normalisiertes Fließkommazahlensystem an, mit welchem sich die folgenden dezimalen Werte gerade noch genau darstellen lassen: jede Verkleinerung von p , e_{\max} oder $-e_{\min}$ muss dazu führen, dass mindestens eine Zahl nicht mehr dargestellt werden kann.

Hinweis: p zählt auch die führende Ziffer.

Tipp: Schreiben Sie sich die normalisierte Binärzahldarstellung der Werte auf, wenn sie für Sie nicht offensichtlich ist.

Werte / *Values*: 2.25 , $\frac{1}{8}$, 0.5 , 16.5 , 2^3

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

$\beta = 2$, $p = 6$, $e_{\min} = -3$, $e_{\max} = 4$.

Provide a smallest possible normalized floating point number system that can still represent the following values exactly: any decrease of the numbers p , e_{\max} or $-e_{\min}$ must imply that at least one of the numbers cannot be represented any more.

Hint: p does also count the leading digit.

Tip: Write down the normalized binary representation of the values, if it is not obvious for you.

8. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche noch!
(und bleibt gesund!)