Übungsstunde

Woche 05

Adel Gavranović
adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Links

- polybox zum Material für die Übungsstunden
- Mail an Assistenten

Follow-up aus vorheriger Übungsstunde

- Die Analogie zwischen math. Summen und for-Loops stimmt, insb. auch für Indizes die gleich sind und wenn `sum` mit 0 initialisiert wurde auch für *leere Summen.*[1]

[1] vgl. W04/S.56 und Wikipedia

Follow-up aus vorheriger Übungsstunde

- Die Analogie zwischen math. Summen und for-Loops stimmt, insb. auch für Indizes die gleich sind und wenn `sum` mit 0 initialisiert wurde auch für *leere Summen.*[1]
- "MacLaurin-Reihe"-Aufgabe war zwar machbar (auch ohne funktionen, subloops oder `std::math`), aber war doch etwas zu schwer

[1] vgl. W04/S.56 und ▸ Wikipedia

Follow-up aus vorheriger Übungsstunde

- Die Analogie zwischen math. Summen und for-Loops stimmt, insb. auch für Indizes die gleich sind und wenn `sum` mit 0 initialisiert wurde auch für *leere Summen.*[1]
- "MacLaurin-Reihe"-Aufgabe war zwar machbar (auch ohne funktionen, subloops oder `std::math`), aber war doch etwas zu schwer
- Schaut euch unbedingt "Lösungs.md" in der Musterlösung zur "Equivalent Resistance"-Aufgabe an (habe ich letzte Woche nicht gesehen)

[1] vgl. W04/S.56 und Wikipedia

Follow-up aus vorheriger Übungsstunde

- Die Analogie zwischen math. Summen und for-Loops stimmt, insb. auch für Indizes die gleich sind und wenn `sum` mit 0 initialisiert wurde auch für *leere Summen.*[1]
- "MacLaurin-Reihe"-Aufgabe war zwar machbar (auch ohne funktionen, subloops oder `std::math`), aber war doch etwas zu schwer
- Schaut euch unbedingt "Lösungs.md" in der Musterlösung zur "Equivalent Resistance"-Aufgabe an (habe ich letzte Woche nicht gesehen)
- Ihr dürft für die Bonusaufgaben jegliche Konzepte und Keywords verwenden, die euch bis zum Abgabedatum beigebracht wurden

[1] vgl. W04/S.56 und ▸ Wikipedia

Follow-up aus vorheriger Übungsstunde

- Die Analogie zwischen math. Summen und for-Loops stimmt, insb. auch für Indizes die gleich sind und wenn `sum` mit 0 initialisiert wurde auch für *leere Summen.*[1]
- "MacLaurin-Reihe"-Aufgabe war zwar machbar (auch ohne funktionen, subloops oder `std::math`), aber war doch etwas zu schwer
- Schaut euch unbedingt "Lösungs.md" in der Musterlösung zur "Equivalent Resistance"-Aufgabe an (habe ich letzte Woche nicht gesehen)
- Ihr dürft für die Bonusaufgaben jegliche Konzepte und Keywords verwenden, die euch bis zum Abgabedatum beigebracht wurden
- Falls ihr Output generieren wollt, der nicht vom Autograder gesehen wird, versuchts mit "std::cerr"

[1] vgl. W04/S.56 und Wikipedia

Kommentare zu [code]expert

- Eine Aufgabe hatte die falsche Einstellung (Code-task statt Text-task) und zeigt eure Submissions fälschlicherweise per default als falsch an, sollte jetzt aber gefixt sein(?)

Kommentare zu [code]expert

- Eine Aufgabe hatte die falsche Einstellung (Code-task statt Text-task) und zeigt eure Submissions fälschlicherweise per default als falsch an, sollte jetzt aber gefixt sein(?)
- Bug(?) zu durchgestrichenen Aufgaben bei Visualisierung zur Bonusaufgabe: Antwort von Team noch ausstehend

Kommentare zu [code]expert

- Eine Aufgabe hatte die falsche Einstellung (Code-task statt Text-task) und zeigt eure Submissions fälschlicherweise per default als falsch an, sollte jetzt aber gefixt sein(?)
- Bug(?) zu durchgestrichenen Aufgaben bei Visualisierung zur Bonusaufgabe: Antwort von Team noch ausstehend
- Weiss jede:r, wie man zwischen markdown und compiled view in den text-tasks schaltet?

[code]expert **E3:T1**

- Bitte nichts schreiben, was nicht Teil der Lösung ist. Und wenn doch, dann unbedingt als Kommentar
- Schnellaufgabe dazu später...

[code]expert E3:T1.5

Two's Complement Binary!

[code]expert **E3:T1.5**

Two's Complement Binary!

1. `0b0001` $\rightarrow 2^0 = 1$
2. `0b0101` $\rightarrow 2^0 + 2^2 = 5$
3. `0b0111` $\rightarrow \ldots = 7$

[code]expert E3:T1.5

Two's Complement Binary!

1. 0b0001
2. 0b0101
3. 0b0111
4. 0b1000 → $2^3 = 8$ → -8
5. 0b1010 → $-2^3 + 2 = -6$
6. 0b1111 → -1

0 0000

[code]expert **E3:T2**

( ))

- Viele von euch haben sehr viele Klammern benutzt, das finde ich toll

[code]expert E3:T2

- Viele von euch haben sehr viele Klammern benutzt, das finde ich toll
- Sehr viele Flüchtigkeitsfehler!
- Sehr viele haben den Unterschied zwischen = und == noch nicht ganz verstanden

assignment

vergleich!

(eval:) b

T/F

a = (b = 5)

[code]expert E3:T2

- Viele von euch haben sehr viele Klammern benutzt, das finde ich toll
- Sehr viele Flüchtigkeitsfehler!
- Sehr viele haben den Unterschied zwischen = und == noch nicht ganz verstanden
- `!a == (a == false)`

[code]expert E3:T2

- Viele von euch haben sehr viele Klammern benutzt, das finde ich toll
- Sehr viele Flüchtigkeitsfehler!
- Sehr viele haben den Unterschied zwischen = und == noch nicht ganz verstanden
- `!a == (a == false)`
- `b == (b == true)`

if(b) if(b==true)

[code]expert **E3:T2**

- Viele von euch haben sehr viele Klammern benutzt, das finde ich toll
- Sehr viele Flüchtigkeitsfehler!
- Sehr viele haben den Unterschied zwischen = und == noch nicht ganz verstanden
- `!a == (a == false)`
- `b == (b == true)`
- *Either a and b are both false or c is true, but not both:*

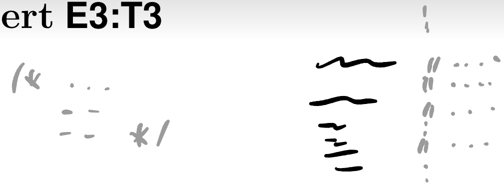
[code]expert **E3:T2**

- Viele von euch haben sehr viele Klammern benutzt, das finde ich toll
- Sehr viele Flüchtigkeitsfehler!
- Sehr viele haben den Unterschied zwischen = und == noch nicht ganz verstanden
- `!a == (a == false)`
- `b == (b == true)`
- *Either a and b are both false or c is true, but not both:* `(!a && !b) != c`
- 5b oder `5*b`?

C++

operator "*"

[code]expert **E3:T3**



- Kommentare am liebsten auf einer Reihe, am besten mit Tabs (insb. nicht ewiglange auf einer Zeile)
- Variablenbenennung (+ Erklärungen zu allen gebrauchten Variablen)
- Benutzt keine Konzepte, die euch noch nicht in der Lektion beigebracht wurden

[code]expert E3:T4

```
// Viele initialisieren mit

int i, n, a, b, x;

// aber ich finde

int i;  // macht dies
int a;  // macht das
int n;  // macht jenes

// besser
```

- Bitte schaut euch meine Verbesserungen zu eurem Code an und versucht sie umzusetzen!

Fragen zu [code]expert eurerseits?

Schnellaufgabe

0, ...

[0, 1]

```
int a;

bool deli = 34467 > 1848 > 8425 > 831 > 8 > 2 > 1;
bool peli = 2 < a < 4;

int doli = deli;
int poli = peli;

std::cout << doli << poli << std::endl;
// What will be the output?
```

0, 1

→ true

Schnellaufgabe

≠ 0, meistens ...

```
int a;

bool deli = 34467 > 1348 > [illegible]425 > [illegible]31 > 8 > 2 > 1;
bool peli = 2 < a < 4;

int doli = deli;
int poli = peli;

std::cout << doli << poli << std::endl;
// What will be the output?
```

Lösung: 01

Lernziele

- ☐ Eine Dezimalzahl (insb. nicht Ganzzahl) in ihre Binärdarstellung umwandeln
- ☐ Wissen, welche Elemente in der Menge $F^*(b, p, e_{min}, e_{max})$ sind und weshalb
- ☐ Arithmetische Operationen innerhalb von $F^*(b, p, e_{min}, e_{max})$ ausführen können

Expressions

Aufgabe

Evaluierer die folgenden Expressions:

1. `5 < 4 < 1`
2. `true > false`

Expressions

Aufgabe

Evaluierer die folgenden Expressions:

1. `5 < 4 < 1`
2. `true > false`

Lösung 1

`5 < 4 < 1`

Expressions

Aufgabe

Evaluierer die folgenden Expressions:

1. `5 < 4 < 1`
2. `true > false`

Lösung 1

```
5 < 4 < 1
(5 < 4) < 1
```

Expressions

Aufgabe

Evaluierer die folgenden Expressions:

1. `5 < 4 < 1`
2. `true > false`

Lösung 1

```
5 < 4 < 1
(5 < 4) < 1
false < 1
```

Expressions

Aufgabe

Evaluierer die folgenden Expressions:

1. `5 < 4 < 1`
2. `true > false`

Lösung 1

```
5 < 4 < 1
(5 < 4) < 1
false < 1
0 < 1
```

Expressions

Aufgabe

Evaluierer die folgenden Expressions:

1. `5 < 4 < 1`
2. `true > false`

Lösung 1

```
5 < 4 < 1
(5 < 4) < 1
false < 1
0 < 1
true
```

Expressions

Aufgabe

Evaluierer 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
```

Expressions

Aufgabe

Evaluierer 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
```

Expressions

Aufgabe

Evaluierer 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
```

Recap: Binary Representation ...aber welche?

Math. Dezimal	Math. Binär	`int`	`unsigned int`

Recap: Binary Representation ...aber welche?

Math. Dezimal	Math. Binär	`int`	`unsigned int`
42_{10}			

Recap: Binary Representation ...aber welche?

Δ Korrigen

Math. Dezimal	Math. Binär	`int`	`unsigned int`
42_{10}	101010_2	`101010`	`101010`

Recap: Binary Representation ...aber welche?

Math. Dezimal	Math. Binär	`int`	`unsigned int`
42_{10}	101010_2	`101010`	`101010`
-42_{10}			

Recap: Binary Representation ...aber welche?

Math. Dezimal	Math. Binär	`int`	`unsigned int`
42_{10}	101010_2	`101010`	`101010`
-42_{10}	-101010_2	`1010110`	nope

Recap: Binary Representation ...aber welche?

Math. Dezimal	Math. Binär	`int`	`unsigned int`
42_{10}	101010_2	`101010`	`101010`
-42_{10}	-101010_2	`1010110`	nope

`int` speichert Zahlen in der *two's complement Representation*.

Binary Arithmetic

Aufgabe

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

Binary Arithmetic

Aufgabe

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

Lösung

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

Binary Arithmetic

Aufgabe

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

Lösung

$a = 4_{10} = 100_2$
$b = 7_{10} = 111_2$

Binary Arithmetic

Aufgabe

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

Lösung

$a = 4_{10} = 100_2$
$b = 7_{10} = 111_2$
$100_2 + 111_2 = 1011_2$

Binary Arithmetic

Aufgabe

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

Lösung

$a = 4_{10} = 100_2$
$b = 7_{10} = 111_2$
$100_2 + 111_2 = 1011_2$
$1011_2 = 11_{10}$

Fragen/Unklarheiten?

Binary Representation

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

Binary Representation

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

Aufgaben

1. 11.01_2 in decimal
2. 101.1_2 in decimal
3. 7.125_{10} in binary
4. 4.375_{10} in binary
5. 1.1 in binary

Binary Representation

$3 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 3.25$

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

Aufgaben

1. $\overset{3}{11}01_2$ in decimal
2. 101.1_2 in decimal
3. 7.125_{10} in binary
4. 4.375_{10} in binary
5. 1.1 in binary

Lösungen

1. $2 + 1 + 0 + \frac{1}{4} = 3.25_{10}$

Binary Representation

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

Aufgaben

1. 11.01_2 in decimal
2. 101.1_2 in decimal
3. 7.125_{10} in binary
4. 4.375_{10} in binary
5. 1.1 in binary

Lösungen

1. $2+1+0+\frac{1}{4}=3.25_{10}$
2. $4+0+1+\frac{1}{2}=5.5_{10}$

Binary Representation

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

Aufgaben

1. 11.01_2 in decimal
2. 101.1_2 in decimal
3. 7.125_{10} in binary
4. 4.375_{10} in binary
5. 1.1 in binary

Lösungen

1. $2 + 1 + 0 + \frac{1}{4} = 3.25_{10}$
2. $4 + 0 + 1 + \frac{1}{2} = 5.5_{10}$
3. $\underbrace{111}_{7}.001_2$

   1/8

Binary Representation

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

Aufgaben

1. 11.01_2 in decimal
2. 101.1_2 in decimal
3. 7.125_{10} in binary
4. 4.375_{10} in binary
5. 1.1 in binary

Lösungen

1. $2 + 1 + 0 + \frac{1}{4} = 3.25_{10}$
2. $4 + 0 + 1 + \frac{1}{2} = 5.5_{10}$
3. 111.001_2
4. 100.011_2

Binary Representation

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

Aufgaben

1. 11.01_2 in decimal
2. 101.1_2 in decimal
3. 7.125_{10} in binary
4. 4.375_{10} in binary
5. 1.1 in binary

Lösungen

1. $2 + 1 + 0 + \frac{1}{4} = 3.25_{10}$
2. $4 + 0 + 1 + \frac{1}{2} = 5.5_{10}$
3. 111.001_2
4. 100.011_2
5. $1.1_{10} = 1.0001100\ldots_2$

$\frac{1}{3} = 0{,}33|\ldots$

Binary Representation

2 | 625

16

Mein Vorgehen

1. Berechne die Ganzzahl vor dem Dezimalpunkt

Binary Representation

Mein Vorgehen

1. Berechne die Ganzzahl vor dem Dezimalpunkt
2. Schreib die Z_2 nieder. Jetzt kommt Z_{10}.REST dran

Binary Representation

Mein Vorgehen

1. Berechne die Ganzzahl vor dem Dezimalpunkt
2. Schreib die Z_2 nieder. Jetzt kommt Z_{10}.REST dran
3. Kann $\frac{1}{2^n}$ von der Zahl Z_{10} abgezogen werden?

Binary Representation

Mein Vorgehen

1. Berechne die Ganzzahl vor dem Dezimalpunkt
2. Schreib die Z_2 nieder. Jetzt kommt Z_{10}.REST dran
3. Kann $\frac{1}{2^n}$ von der Zahl Z_{10} abgezogen werden?
   falls ja, subtrahiere $\frac{1}{2^n}$ von Z_{10} und füge eine 1 am Ende von Z_2 hinzu

Binary Representation

$2^{-1} = \frac{1}{2}$

Mein Vorgehen

1. Berechne die Ganzzahl vor dem Dezimalpunkt
2. Schreib die Z_2 nieder. Jetzt kommt Z_{10}.REST dran

   $n = 1$
3. Kann $\frac{1}{2^n}$ von der Zahl Z_{10} abgezogen werden?
   falls ja, subtrahiere $\boxed{\frac{1}{2^n}}$ von Z_{10} und füge eine 1 am Ende von Z_2 hinzu
   falls nein, füge eine 0 am Ende von Z_2 hinzu

Binary Representation

Mein Vorgehen

1. Berechne die Ganzzahl vor dem Dezimalpunkt
2. Schreib die Z_2 nieder. Jetzt kommt Z_{10}.REST dran
3. Kann $\frac{1}{2^n}$ von der Zahl Z_{10} abgezogen werden?
   falls ja, subtrahiere $\frac{1}{2^n}$ von Z_{10} und füge eine 1 am Ende von Z_2 hinzu
   falls nein, füge eine 0 am Ende von Z_2 hinzu
4. falls $\boxed{Z_{10}}$ jetzt bei 0 angekommen ist, überprüfe deine Lösung und sonst, führe diesen Algorithmus mit n++ fort

Fragen/Unklarheiten?

Normalized Floating Point Systems

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

$*$ normalisiert ($b_0 \neq 0$)

Normalized Floating Point Systems

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

$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis

Normalized Floating Point Systems

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

$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)

Normalized Floating Point Systems

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

$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)
e_{min}	kleinstmöglicher Exponent

Normalized Floating Point Systems

$F^*(\beta, p, e_{min}, e_{max})$	
$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)
e_{min}	kleinstmöglicher Exponent
e_{max}	grösstmöglicher Exponent

Normalized Floating Point Systems

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

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

d_0

...beschreibt Zahlen der Form:

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

Normalized Floating Point Systems

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

$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)
e_{min}	kleinstmöglicher Exponent
e_{max}	grösstmöglicher Exponent

... beschreibt Zahlen der Form:

$$\pm d_0.d_1 d_2 d_3 \dots d_{p-1} \cdot \beta^e$$

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

Normalized Floating Point Systems

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

$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)
e_{min}	kleinstmöglicher Exponent
e_{max}	grösstmöglicher Exponent

...beschreibt Zahlen der Form:

2 : 0, 1
10 : 0, 1 . .. 9

$$\pm d_0.d_1 d_2 d_3 \dots d_{p-1} \cdot \beta^e$$

$d_i \in \{0, \dots, b-1\}$
$d_0 \neq 0$

Normalized Floating Point Systems

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

$*$	normalisiert ($b_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)
e_{min}	kleinstmöglicher Exponent
e_{max}	grösstmöglicher Exponent

…beschreibt Zahlen der Form:

$$\pm d_0.d_1 d_2 d_3 \ldots d_{p-1} \cdot \beta^e$$

$d_i \in \{0, \ldots, b-1\}$
$d_0 \neq 0$
$e \in [e_{min}, e_{max}]$

Fragen/Unklarheiten?

Übungen

Übungen

Are the following numbers in the set $F^*(2, 4, -2, 2)$?

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

$$0.000 \cdot 2^1 = 0_{10}$$
$$1.000 \cdot 2^1 = 2_{10}$$
$$1.001 \cdot 2^{-1} = 0.5625_{10}$$
$$1.000\boxed{1} \cdot 2^{-1} = 0.53125_{10}$$
$$1.111 \cdot 2^{-2} = 0.46875_{10}$$
$$1.111 \cdot 2^5 = 60_{10}$$

Übungen

Übungen

Are the following numbers in the set $F^*(2, 4, -2, 2)$?

$$
\begin{aligned}
0.000 \cdot 2^1 &= 0_{10} \\
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.0001 \cdot 2^{-1} &= 0.53125_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10} \\
1.111 \cdot 2^5 &= 60_{10}
\end{aligned}
$$

Lösungen

in F^*

$$
\begin{aligned}
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10}
\end{aligned}
$$

Übungen

Übungen

Are the following numbers in the set $F^*(2, 4, -2, 2)$?

$$
\begin{aligned}
0.000 \cdot 2^1 &= 0_{10} \\
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.0001 \cdot 2^{-1} &= 0.53125_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10} \\
1.111 \cdot 2^5 &= 60_{10}
\end{aligned}
$$

Lösungen

in F^*

$$
\begin{aligned}
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10}
\end{aligned}
$$

nicht in F^*

$0.000 \cdot 2^1$

Übungen

Übungen

Are the following numbers in the set $F^*(2, 4, -2, 2)$?

$$
\begin{aligned}
0.000 \cdot 2^1 &= 0_{10} \\
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.0001 \cdot 2^{-1} &= 0.53125_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10} \\
1.111 \cdot 2^5 &= 60_{10}
\end{aligned}
$$

Lösungen

in F^*

$$
\begin{aligned}
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10}
\end{aligned}
$$

nicht in F^*

$0.000 \cdot 2^1$ nicht ”normalizable”

$1.0001 \cdot 2^{-1}$

Übungen

Übungen

Are the following numbers in the set $F^*(2, 4, -2, 2)$?

$$
\begin{aligned}
0.000 \cdot 2^1 &= 0_{10} \\
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.0001 \cdot 2^{-1} &= 0.53125_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10} \\
1.111 \cdot 2^5 &= 60_{10}
\end{aligned}
$$

Lösungen

in F^*

$$
\begin{aligned}
1.000 \cdot 2^1 &= 2_{10} \\
1.001 \cdot 2^{-1} &= 0.5625_{10} \\
1.111 \cdot 2^{-2} &= 0.46875_{10}
\end{aligned}
$$

nicht in F^*

$0.000 \cdot 2^1$	nicht "normalizable"
$1.0001 \cdot 2^{-1}$	$5 > p = 4$
$1.111 \cdot 2^5$	

Übungen

?: Wenn 1.0 im gegebenem F* mit p=4 ist, ist dann auch 1.000000 in diesem F*?
!: Ja, da die "trailing zeros" hier irrelevant sind und die Präzision nicht ändern (anders als im Naturwissenschaftl. Kontext, wo man mit den Nullen eine grössere Präzision implizieren würde.)
→ mehr dazu in der nächsten Stunde

Übungen

Are the following numbers in the set $F^*(2,4,-2,2)$?

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

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

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

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

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

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

Lösungen

in F^*

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

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

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

nicht in F^*

$0.000 \cdot 2^1$ nicht "normalizable"

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

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

Fragen/Unklarheiten?

mehr Übungen

$$1{,}111 \cdot 2^2 = \overset{7}{111}{,}\overset{0{,}5}{1} = 7{,}5$$

$$-1{,}111 \cdot 2^2 = \ldots = -7{,}5$$

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

$$1{,}000 \cdot 2^{-2}$$

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

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

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

Lösung

grösste:

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

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

Lösung

grösste: $1.111 \cdot 2^2 = 7.5_{10}$
kleinste:

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

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

Lösung

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

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

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

Lösung

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

Normalized Floating Point Systems

Lösung

0

Für einen gefixten 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.

Normalized Floating Point Systems

Lösung

Für einen gefixten 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.

Trick

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

grösste:	$\overbrace{1.11\ldots1}^{p} \cdot 2^{e_{max}}$
kleinste:	$-$~~Largest~~ grösste
kleinste > 0:	$\underbrace{1.00\ldots0}_{p} \cdot 2^{e_{min}}$

Fragen/Unklarheiten?

Arithmetik in F^*

11 · 2

750 : 10

Floats addieren

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

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$
$1.001_2 + 1001.01_2$ (bereits gleicher Exponent $(\cdot 2^0)$)

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$
$1.001_2 + 1001.01_2$ (bereits gleicher Exponent ($\cdot 2^0$))
1010.011 (Addition wie in der 2. Klasse)

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$
$1.001_2 + 1001.01_2$ (bereits gleicher Exponent $(\cdot 2^0)$)
1010.011 (Addition wie in der 2. Klasse)
$1.010011 \cdot 2^3$ Renormalisierung, e anpassen, anpassen für p
6

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$
$1.001_2 + 1001.01_2$ (bereits gleicher Exponent ($\cdot 2^0$))
1010.011 (Addition wie in der 2. Klasse)
$1.010011 \cdot 2^3$ Renormalisierung, e anpassen, anpassen für p
$1.01010 \cdot 2^3$ Runden: 1 rauf, 0 runter, und übertragen

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$
$1.001_2 + 1001.01_2$ (bereits gleicher Exponent ($\cdot 2^0$))
1010.011 (Addition wie in der 2. Klasse)
$1.010011 \cdot 2^3$ Renormalisierung, e anpassen, anpassen für p
$1.01010 \cdot 2^3$ Runden: 1 rauf, 0 runter, und übertragen
$1.01010_2 \cdot 2_2^3 = 1010.10_2 = 10.5_{10}$

Arithmetik in F^*

Floats addieren

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

Beispiel

$F^*(2, 6, -2, 3)$
$1.125_{10} + 9.25_{10}$
$1.001_2 + 1001.01_2$ (bereits gleicher Exponent $(\cdot 2^0)$)
1010.011 (Addition wie in der 2. Klasse)
$1.010011 \cdot 2^3$ Renormalisierung, e anpassen, anpassen für p
$1.01010 \cdot 2^3$ Runden: 1 rauf, 0 runter, und übertragen
$1.01010_2 \cdot 2_2^3 = 1010.10_2 = 10.5_{10} \neq 10.375_{10}$

Wieso 10.5 und nicht 10.375?

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.

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

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. Bring beide auf den gleichen Exponenten, sagen wir -1
   $1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$

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. Bring beide auf den gleichen Exponenten, sagen wir -1
   $1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$
2. Addier sie in der Binärdarstellung: $10.0001 \cdot 2^{-1}$

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. Bring beide auf den gleichen Exponenten, sagen wir -1
   $1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$
2. Addier sie in der Binärdarstellung: $10.0001 \cdot 2^{-1}$
3. Renormalisiere: $1.00001 \cdot 2^{0}$

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. Bring beide auf den gleichen Exponenten, sagen wir -1
   $1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$
2. Addier sie in der Binärdarstellung: $10.0001 \cdot 2^{-1}$
3. Renormalisiere: $1.00001 \cdot 2^{0}$
4. Runde: $1.000 \cdot 2^{0}$

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. Bring beide auf den gleichen Exponenten, sagen wir -1
   $1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$
2. Addier sie in der Binärdarstellung: $10.0001 \cdot 2^{-1}$
3. Renormalisiere: $1.00001 \cdot 2^{0}$
4. Runde: $1.000 \cdot 2^{0} = 1_{10}$

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. Bring beide auf den gleichen Exponenten, sagen wir -1
   $1.001 \cdot 2^{-1} + 0.1111 \cdot 2^{-1}$
2. Addier 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?

Floating Point guidelines

+ kleine Demo zu schönerem Code, falls Zeit vorhanden

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";
```

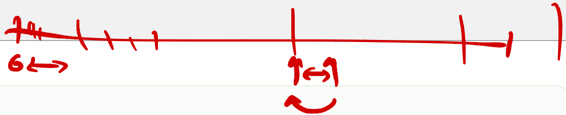
Problem:

`1.05f` not representable

$$1.05 \quad = \overbrace{1.00001100110011001100110011001\ldots}^{24\text{bit}} \cdot 2^0$$

(rounding) $\rightarrow 1.049999995231\ldots = 1.00001100110011001100110 \cdot 2^0$

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**!»

Problem:

Significand too short

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**!»

Problem:

Significand too short

Example:

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

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

$$
\begin{array}{rcl}
67108864 & = & \overbrace{1.000000000000000000000000}^{24\text{bit}} \quad \cdot 2^{26} \\
+1 & = & 0.00000000000000000000000001 \cdot 2^{26} \\
\hline
67108865 & = & 1.00000000000000000000000001 \cdot 2^{26}
\end{array}
$$

Guideline 2 – Example

Guideline 2:

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

Problem:

Significand too short

Example:

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

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

$$\begin{aligned}
67108864 &= \overbrace{1.000000000000000000000000}^{\text{24bit}} \quad \cdot 2^{26} \\
+1 &= 0.0000000000000000000000001 \cdot 2^{26} \\
\hline
67108865 &= 1.0000000000000000000000001 \cdot 2^{26} \\
\text{(rounding)} \rightarrow 67108864 &= 1.000000000000000000000000 \quad \cdot 2^{26}
\end{aligned}$$

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 $\mathrm{x}_{n+1} = 6x_n - 1$
- Computing some sequences for given x_0:

Guideline 3 – Example

Guideline 3:

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

Example:

- Consider sequence $\mathrm{x}_{n+1} = 6x_n - 1$
- Computing some sequences for given x_0:
    - e.g. $x_0 = 1$ → $x_1 = 5, \quad x_2 = 29, \quad x_3 = 173, \quad \ldots$

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. $x_0 = 1$ → $x_1 = 5, \quad x_2 = 29, \quad x_3 = 173, \quad \ldots$
  - e.g. $x_0 = 0.2$ → $x_1 = 0.2, \quad x_2 = 0.2, \quad x_3 = 0.2, \quad \ldots$

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. $x_0 = 1$ → $x_1 = 5, \quad x_2 = 29, \quad x_3 = 173, \quad \ldots$
  - e.g. $x_0 = 0.2$ → $x_1 = 0.2, \quad x_2 = 0.2, \quad x_3 = 0.2, \quad \ldots$

C++ claims

$\boldsymbol{x_{14} \approx 622.982}$

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:

$$x_1 = 0.20000004768\ldots$$
$$\mathrm{x}_2 = 0.20000028610\ldots$$
$$x_3 = 0.20000171661\ldots$$
$$\vdots$$

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:

$$x_1 = 0.2000000\mathbf{4768}\ldots$$
$$\mathrm{x}_2 = 0.200000\mathbf{28610}\ldots$$
$$x_3 = 0.20000\mathbf{171661}\ldots$$
$$\vdots$$

Note how error increases!

Floating Point Numbers Vergleichen

Kurzgesagt: nicht auf Gleichheit prüfen
lieber auf ”innerhalb der Toleranz” prüfen

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

Comparing FP-Numbers

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.)

Prüfungsfrage

EURE PRÜFUNG WIRD ANDERS - VERGESST DAS NICHT

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.

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.

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

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

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

Prüfungsfrage

EURE PRÜFUNG WIRD ANDERS - VERGESST DAS NICHT

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.

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.

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

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

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

Allgemeine Fragen?

Bis zum nächsten Mal

- ☐ macht eure Hausaufgaben
- ☐ bleibt gesund
- ☐ geniesst die Sonne solange ihr noch könnt, ihr werdet sie in der Lernphase vermissen