ETH zürich

Exercise Session W05

Computer Science (CSE) – AS 23

Overview

Today's Agenda

Follow-up
Feedback on **code** expert
Objectives
Repetition
Binary Representation
Normalized Floating Point Systems
Floating Point Guidelines
Exam Question
Outro

n.ethz.ch/~agavranovic

1. Follow-up

Follow-up from last exercise session

- Hope you all liked Lily!

Question from last Exercise Session

TLDL; l → r

TRUE

shortcircuits

```
int x = 1;
int y = 1;

bool EXPRESSION = (++x == 2) || (++x/0) || ((y-- == 0) && (++y == 0));
// Value of EXPRESSION?

std::cout << EXPRESSION << std::endl;
// Output?
```

2. Feedback on **code** expert

General things regarding **code** expert

General things regarding **code** expert

- Corrections are still being made (Some programming tasks still outstanding)

General things regarding **code** expert

- Corrections are still being made (Some programming tasks still outstanding)
- You're allowed to use any concept that was introduced before the deadline (unless it makes the exercise trivially easy)
  - If you're unsure: send me an email

General things regarding **code** expert

- Corrections are still being made (Some programming tasks still outstanding)
- You're allowed to use any concept that was introduced before the deadline (unless it makes the exercise trivially easy)
  - If you're unsure: send me an email
- Please make sure to read the feedback I give you and try your best to implement it for the next submissions

How to Comment

How to Comment

- Rule of Thumb: If you’re just stating what is written in code then delete the comment
- Comments should add context
- Give your variables descriptive names and use comments to guide the reader through your thought process
- Alternatively, write your code so well that it needs no comments

How to Comment

```
// S a to value 6
const unsigned int a = 6;

// Output "Hello World!\n" to the console.
std::cout << "Hello World!";
```

How to Comment

```
// S a to value 6
const unsigned int a = 6;

// Output "Hello World!\n" to the console.
std::cout << "Hello World!";
```

Better:

```
// resistance of component A to 6 Ohm
const unsigned int resistance_A = 6;

// Greet World
std::cout << "Hello World!";
```

3. Objectives

Objectives

- ☐ Be able to convert a decimal (non-integer) number into its binary representation
- ☐ Be able to compute the elemts of the set $F^*(b, p, e_{\text{min}}, e_{\text{max}})$
- ☐ Be able to perform arithmetic operations in set $F^*(b, p, e_{\text{min}}, e_{\text{max}})$

4. Repetition

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
  f
```

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 ...but which one?

Math. Dec	Math. Bin	`int`	`unsigned int`

Recap: Binary Representation ...but which one?

Math. Dec	Math. Bin	`int`	`unsigned int`
42_{10}			

Recap: Binary Representation ...but which one?

Math. Dec	Math. Bin	`int`	`unsigned int`
42_{10}	101010_2	`101010`	`101010`

Recap: Binary Representation ...but which one?

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

Recap: Binary Representation ...but which one?

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

Recap: Binary Representation ...but which one?

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

`int` saves numbers in *two's complement representation*.

Binary Arithmetic

Tasks

1. Convert the whole numbers $a = 4$ and $b = 7$ into their binary representations (not two's complement)
2. Add the two (in their binary representation)
3. Convert the result back into decimal representation

Binary Arithmetic

Tasks

1. Convert the whole numbers $a = 4$ and $b = 7$ into their binary representations (not two's complement)
2. Add the two (in their binary representation)
3. Convert the result back into decimal representation

Lösung

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

Binary Arithmetic

Tasks

1. Convert the whole numbers $a = 4$ and $b = 7$ into their binary representations (not two's complement)
2. Add the two (in their binary representation)
3. Convert the result back into decimal representation

Lösung

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

Binary Arithmetic

Tasks

1. Convert the whole numbers $a = 4$ and $b = 7$ into their binary representations (not two's complement)
2. Add the two (in their binary representation)
3. Convert the result back into decimal representation

Lösung

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

$8 + 2 + 1 = 11$

Binary Arithmetic

Tasks

1. Convert the whole numbers $a = 4$ and $b = 7$ into their binary representations (not two's complement)
2. Add the two (in their binary representation)
3. Convert the result back into decimal representation

Lösung

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

Questions?

5. Binary Representation

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

Exercises

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_{10} in binary

Decimal Number

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

Exercises

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

Solutions

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

Exercises

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

Solutions

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

Exercises

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

Solutions

1. $2 + 1 + 0 + \frac{1}{4} = 3.25_{10}$
2. $4 + 0 + 1 + \frac{1}{2} = 5.5_{10}$
3. 111.001_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	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

Exercises

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

Solutions

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

Exercises

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

Solutions

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. $1.1_{10} = 1.000110\ldots_2$

Binary Representation

My personal approach

1. Calculate the nunmber in front of the decimal point

Binary Representation

My personal approach

1. Calculate the nunmber in front of the decimal point
2. Write out the Z_2. Now to the Z_{10}.REST

Binary Representation

My personal approach

1. Calculate the nunmber in front of the decimal point
2. Write out the Z_2. Now to the Z_{10}.REST
3. Can $\frac{1}{2^n}$ be deducted from Z_{10}?

Binary Representation

My personal approach

1. Calculate the nunmber in front of the decimal point
2. Write out the Z_2. Now to the Z_{10}.REST
3. Can $\frac{1}{2^n}$ be deducted from Z_{10}?
   If so, then subtract $\frac{1}{2^n}$ from Z_{10} and add a 1 to the end of Z_2

Binary Representation

My personal approach

1. Calculate the nunmber in front of the decimal point
2. Write out the Z_2. Now to the Z_{10}.REST
3. Can $\frac{1}{2^n}$ be deducted from Z_{10}?
   If so, then subtract $\frac{1}{2^n}$ from Z_{10} and add a 1 to the end of Z_2
   if not, add a 0 to the end of Z_2

Binary Representation

My personal approach

1. Calculate the nunmber in front of the decimal point
2. Write out the Z_2. Now to the Z_{10}.REST
3. Can $\frac{1}{2^n}$ be deducted from Z_{10}?
   If so, then subtract $\frac{1}{2^n}$ from Z_{10} and add a 1 to the end of Z_2
   if not, add a 0 to the end of Z_2
4. if Z_{10} reached 0, you're done
   if not, proceed $n + 1$

Questions?

6. Normalized Floating Point Systems

Normalized Floating Point Systems

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

* normalisiert ($d_0 \neq 0$)

Normalized Floating Point Systems

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

$*$	normalisiert ($d_0 \neq 0$)
$\beta \geq 2$	Basis

Normalized Floating Point Systems

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

$*$	normalisiert ($d_0 \neq 0$)
$\beta \geq 2$	Basis
$p \geq 1$	Präzision (Anzahl Ziffern)

Normalized Floating Point Systems

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

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

Normalized Floating Point Systems

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

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

Normalized Floating Point Systems

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

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

$e \in \mathbb{Z}$

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

…beschreibt Zahlen der Form:

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

Normalized Floating Point Systems

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

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

…beschreibt Zahlen der Form:

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

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

Normalized Floating Point Systems

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

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

...beschreibt Zahlen der Form:

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

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

Normalized Floating Point Systems

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

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

…beschreibt Zahlen der Form:

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

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

Questions?

Übungen

Übungen

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

- ✗ $0.000 \cdot 2^1 = 0_{10}$
- ✓ $1.000 \cdot 2^1 = 2_{10}$
- ✓ $1.001 \cdot 2^{-1} = 0.5625_{10}$
- ✗ $1.0001 \cdot 2^{-1} = 0.53125_{10}$
- ✓ $1.111 \cdot 2^{-2} = 0.46875_{10}$
- ✗ $1.111 \cdot 2^5 = 60_{10}$

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

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

...beschreibt Zahlen der Form:

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

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

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

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"

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

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

Questions?

mehr Übungen

Aufgabe

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

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

1. $1.111 \cdot 2^{2} = 7{,}5_{10}$
2. $-1.111 \cdot 2^{2} = -7{,}5_{10}$
3. $1.000 \cdot 2^{-2} = 0{,}25$

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

160

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

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

grösste:	$1.11\ldots1 \cdot 2^{e_{\text{max}}}$
kleinste:	$-$Largest
kleinste > 0:	$1.00\ldots0 \cdot 2^{e_{\text{min}}}$

Questions?

Arithmetik in F^*

Floats addieren

1. Beide zum gleichen Exponenten umformen
2. In Binärdarstellung addieren
3. Summe renormalisieren   1. . . .
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

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

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
$1.01010_2 \cdot 2^3_2 = 1010.10_2 = 10.5_{10} \neq 10.375_{10}$

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

Floats addieren

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

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

Questions?

7. Floating Point Guidelines

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

$$1.05 = \overbrace{1.00000110011001100110011}^{24\text{bit}}001\ldots \cdot 2^0$$

(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";
```

$$
\begin{array}{rcl}
67108864 & = & \overbrace{1.00000000000000000000000}^{\text{24bit}} \quad \cdot 2^{26} \\
+1 & = & 0.00000000000000000000000001 \cdot 2^{26} \\
\hline
67108865 & = & 1.00000000000000000000000001 \cdot 2^{26}
\end{array}
$$

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";
```

24bit

$$\begin{aligned} 67108864 &= \overbrace{1.00000000000000000000000}^{\text{24bit}} \cdot 2^{26} \\ +1 &= 0.000000000000000000000000001 \cdot 2^{26} \\ \hline 67108865 &= 1.000000000000000000000000001 \cdot 2^{26} \\ \text{(rounding)} \rightarrow 67108864 &= 1.00000000000000000000000 \cdot 2^{26} \end{aligned}$$

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

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

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

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!

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

```
// 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.)

Exam Question

YOUR EXAM MIGHT DIFFER

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 .

Exam Question

YOUR EXAM MIGHT DIFFER

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*

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

9. Outro

General Questions?

Till next time!

Cheers!