Informatik I - Übung 4

Pascal Schärli

pascscha@student.ethz.ch

15.03.2019

ETH zürich

Was gibts heute?

- Nachbesprechung
- Best-Of Vorlesung:
  - Floats
  - Funktionen
- Vorbesprechung

Nachbesprechung

Pascal Schärli *15.03.2019*

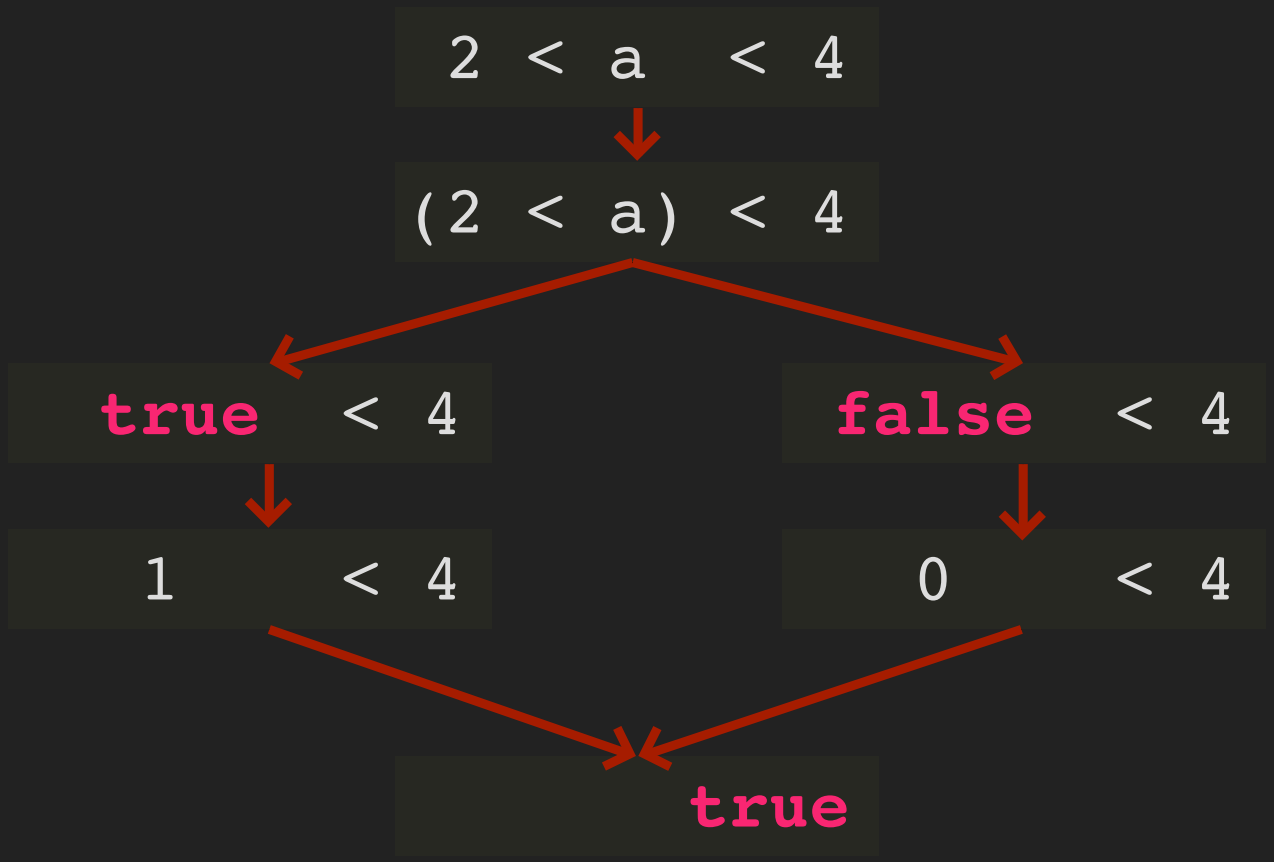
Bild: look again by corpus delicti from the Noun Project

ETH zürich

1. True or False?

```
   8 > 4   > 2   > 1
((8 > 4) > 2) > 1
( true   > 2) > 1
(    1   > 2) > 1
        false  > 1
           0   > 1
               false
```

2. True or False? (a ist int)



Vergleich vs Zuweisung

`==` Vergleich

- Wird gebraucht um die Grösse zu vergleichen
- Verändert die Variablen nicht

Beispiel:

```
if(a==b){
    //...
}
```

`=` Zuweisung

- Wird gebraucht um Variablen neue Werte zuzuweisen
- Der Wert vom rechten Operand wird im linken Operand gespeichert.

Beispiel:

```
int a;
int b = 5;
a = b;
a = 7;
```

Booleans

```
a == false || b == true)
```

```
!a || b
```

Layout Matters!

```
#include <iostream>
int main(){
int n;
std::cin >> n; for(int i = 0; i < n; i++)
if(i % 2 == 0)

std::cout << i / 2 << std::endl;
        else

std::cout << i * 2 << std::endl;
return 0;}
```

```
#include <iostream>

int main(){

    int n;
    std::cin >> n;

    for(int i = 0; i < n; i++){
        if(i % 2 == 0){
            std::cout << i / 2 << std::endl
        }
        else{
            std::cout << i * 2 << std::endl
        }
    }

    return 0;
}
```

Best of Vorlesung

Warm-Up

Expression	Dezimal	Binär
a	`4`	`0b100`
b	`7`	`0b111`
a + b	`11`	`0b1011`

Floats als Binärzahlen

```
ab.cd => 2*a + 1*b + 1/2 * c + 1/4 * d
```

```
00.00 => 0
00.01 => 1/4
00.10 => 1/2
00.11 => 3/4
01.00 => 1
01.01 => 1 + 1/4
01.10 => 1 + 1/2
01.11 => 1 + 3/4
```

```
10.00 => 2
10.01 => 2 + 1/4
10.10 => 2 + 1/2
10.11 => 2 + 3/4
11.00 => 3
11.01 => 3 + 1/4
11.10 => 3 + 1/2
11.11 => 3 + 3/4
```

Floats als Binärzahlen

x_i	b_i	$x - b_i$	$x_{i+1} = 2 * (x - b_i)$
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

$$1.9 \rightarrow 1.1\overline{1100}$$

Floats als Binärzahlen

Warum kann man 0.1 als Binärzahl nicht endlich darstellen, aber als Dezimalzahl schon?

- Um 0.1 oder $\frac{1}{10}$ als endliche Zahl zu notieren, benötigt man die beiden Primzahlen 5 und 2 (weil $10 = 5 * 2$).
- Das Dezimale Zahlensystem ist genau auf diesen beiden Zahlen basiert, daher geht es.
- Das Binäre Zahlensystem hat nur die Zahl 2, daher kann man sie damit nicht endlich darstellen.

Floats als Binärzahlen

Berechne die Binärdarstellung der folgenden Dezimalzahlen:

`0.25` ⇨ `0.01`

`11.1` ⇨ `1011.00011`

Algorithmus:

$b_i \rightarrow$ Binärziffer

$x_i \rightarrow$ Dezimalzahl

Zuerst binärzahl vor Komma normal ausrechnen.

Danach folgende Schritte wiederholen:

1. b_i : Zahl vor Komma
2. $x_{i+1} = 2 * (x_i - b_i)$

FP - Systeme

$F^*(b, p, e_{min}, e_{max})$ ⇨ $\pm b_0.b_1 b_2 \cdots b_{p-1} * b^e$

$b_i \in \{0, \ldots b-1\}$ *(Basis des Zahlensystems)*

$b_0 ! = 0$ *(Normalisiert)*

$p \rightarrow$ Anzah Ziffern

$e \in \{e_{min}, e_{min}+1, \ldots, e_{max}\}$ *(Bereich für Exponent)*

FP - Systeme

Sind diese Zahlen sind im folgenden Zahlensystem enthalten: $F^*(2, 4, -2, 2)$? $\pm b_0.b_1 b_2 \cdots b_{p-1} * b^e$

`1.111 * 2^2`	✓	
`0.000 * 2^1`	✗	normalisiert $\rightarrow$ Zahl startet nicht mit 0
`1.001 * 2^-1`	✓	
`1.0001 * 2^-1`	✗	zu viele Nachkommastellen
`1.111 * 2^5`	✗	Exponent ist nicht zwischen -2 und 2.
`1.000 * 2^1`	✓	

FP - Systeme

Was sind die folgenden Zahlen in $F^*(2, 4, -2, 2)$?

$$\pm b_0.b_1 b_2 \cdots b_{p-1} * b^e$$

die grösste

```
1.111 * 2^2 ->  7.5
```

die kleinste

```
-1.111 * 2^2 -> -7.5
```

die kleinste nicht-negative.

```
1.000 * 2^-2 -> 0.25
```

Rechnen in FP-Systemen

Wie kann man Zahlen in einem FP-System addieren?

- Beide Zahlen auf den selben Exponenten bringen
- Binärzahlen addieren (wie schriftliche Addition)
- Summe re-Normalisieren (1. . . .)
- Runden falls nötig

Rechnen in FP-Systemen

```
1.001 * 2^-1 + 1.111 * 2^-2
```

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

```
    1.001   * 2^-1
+   0.1111  * 2^-1
-----------------
= 10.0001  * 2^-1

= 1.00001 * 2^0

=> 1.000   * 2^0
```

Algorithmus:

1. Beide Zahlen auf den selben Exponenten bringen
2. Binärzahlen addieren (wie schriftliche Addition)
3. Summe re-Normalisieren (1.\...)
4. **Runden falls nötig**

$\rightarrow$ Resultat: 1 (Exakt: 1.03125)

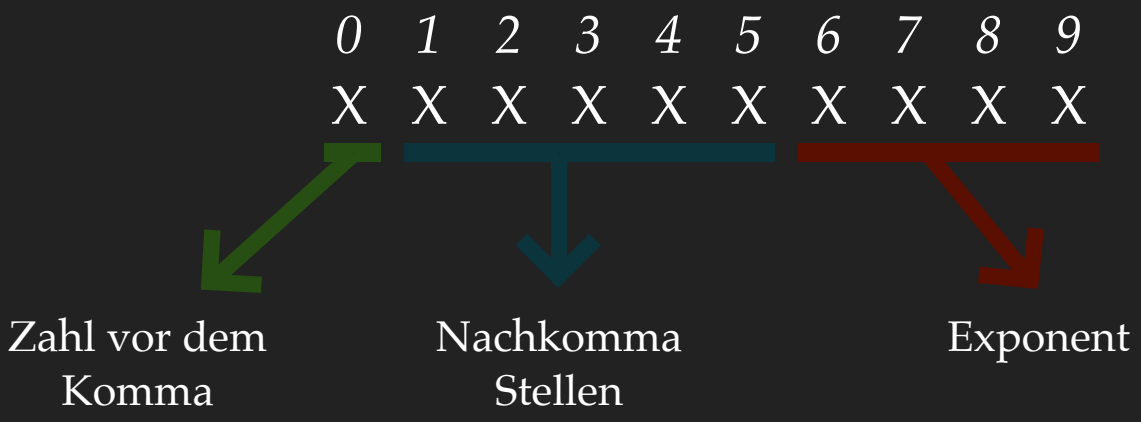
Unser eigener Float-Datentyp

Wir haben 10 Bits zur Verfügung.
Wie können wir damit einen
eigenen Fliesskomma-Datentyp
erstellen?

Unser eigener Float-Datentyp

Versuch 1

Wir versuchen die Wissenschaftliche Notation (e.g. $2.73 \cdot 10^{12}$) zu imitieren



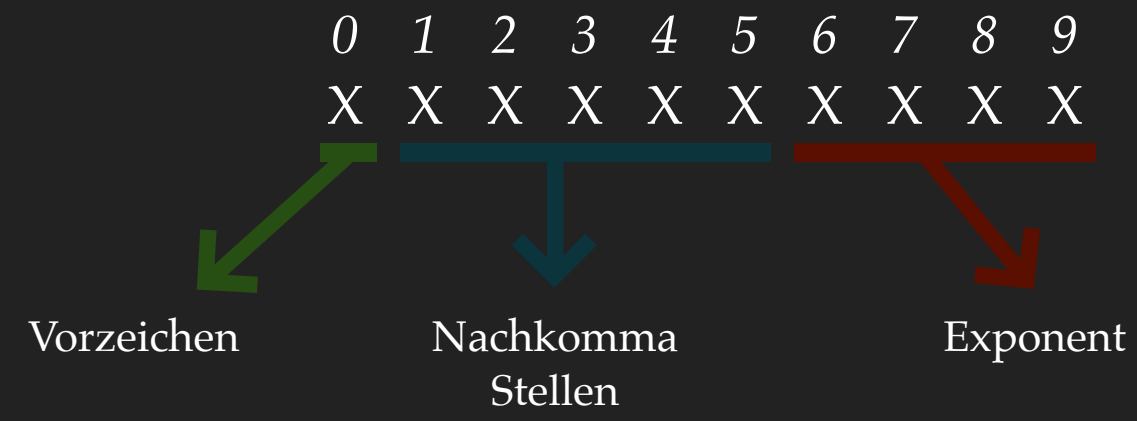
Einige Zahlen in unserem Sytem:

- grösste Zahl: $1.11111 \cdot 2^{15} = 64512$
- kleinste Zahl: $0.00000 * 2^{0} = 0$
- kleinste Positive Zahl: $0.00000 * 2^{0} = 0$

Unser eigener Float-Datentyp

Versuch 2

Wir klassifizieren unser System als $F^*(b, p, e_{min}, e_{max})$ System. Da das erste Bit immer 1 ist, verwenden wir es als Vorzeichen.



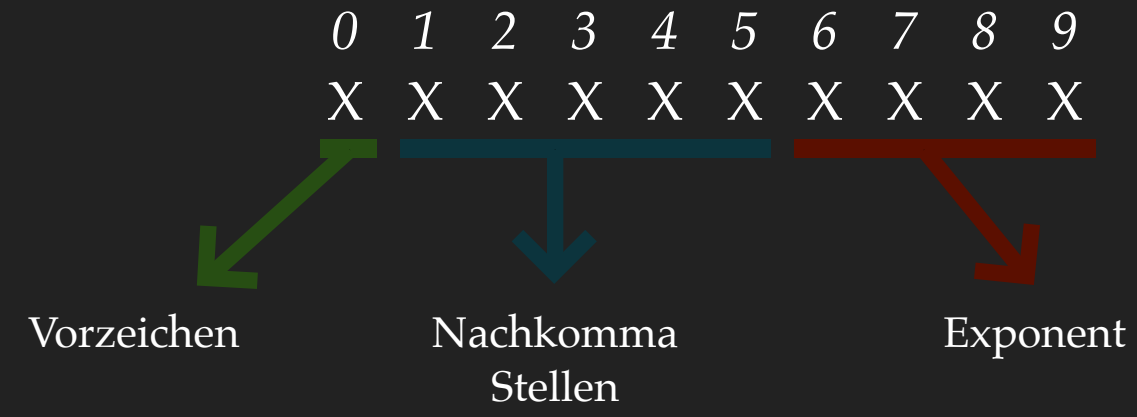
Einige Zahlen in unserem Sytem:

- grösste Zahl: $1.11111 \cdot 2^{15} = 64512$
- kleinste Zahl: $-1.11111 * 2^{15} = -64512$
- kleinste Positive Zahl: $1.00000 * 2^0 = 1$

Unser eigener Float-Datentyp

Versuch 3

Um Fliesskommazahlen zu Erhalten müssen wir negative Nachkommazahlen haben. Wir ziehen daher dem Exponent jeweils 8 ab.



Einige Zahlen in unserem Sytem:

- grösste Zahl: $1.11111 \cdot 2^7 = 252$
- kleinste Zahl: $-1.11111 * 2^7 = -252$
- kleinste Positive Zahl: $1.00000 * 2^{-8} = 0.00390625$

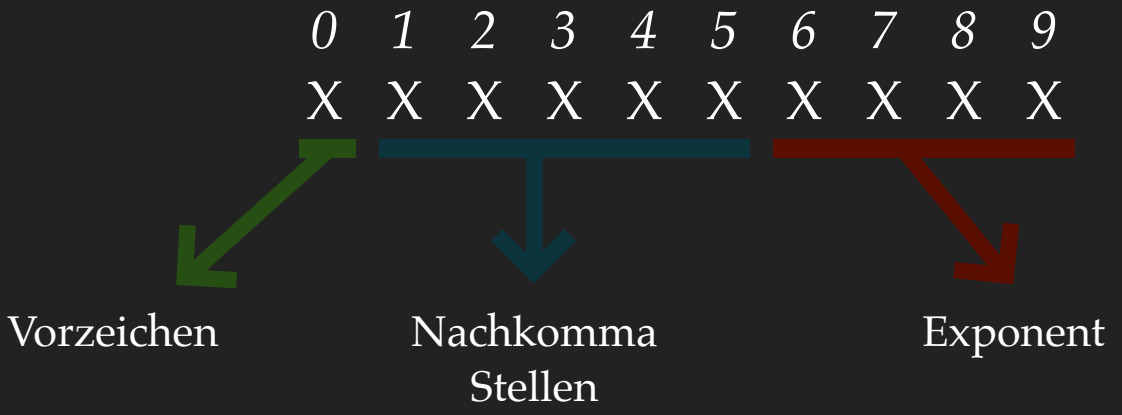
Unser eigener Float-Datentyp

Versuch 4

Es gibt keine 0 in unserem System! Wir definieren also, dass wir beim Exponent -8 eine Spezielle Zahl haben.

Bits 1-5:

- $0000 \rightarrow 0$
- $0001 \rightarrow +\infty$
- $0010 \rightarrow -\infty$
- $0011 \rightarrow \text{NaN}$



Einige Zahlen in unserem Sytem:

- grösste Zahl: ∞
- kleinste Zahl: $-\infty$
- kleinste Positive Zahl: 0

Floats: Guidelines

«Prüft **nie** ob zwei Fliesskommazahlen gleich sind, wenn mindestens eine vorher **gerundet** wurde!»

```
float a = 0.2f;
if (10*a == 2.0f){
        std::cout << "Equal" << std::endl;
}
else{
        std::cout << "Not Equal" << std::endl;
}
```

```
Not Equal
```

Floats: Guidelines

«**Vermeidet** die Addition von Zahlen mit **extremen Unterschieden** in der **Grösse**!»

```
float a = 67108864.0f + 1.0f;
if (a > 67108864.0f){
        std::cout << "Greater" << std::endl;
}
else{
        std::cout << "Not Greater" << std::endl;
}
```

```
Not Greater
```

(Rundungsfehler)

```
   67108864 = 1.00000000000000000000000   * 2^26
         +1 = 0.00000000000000000000000001 * 2^26
  ----------------------------------------------
 = 67108865 = 1.00000000000000000000000001 * 2^26
=> 67108864 = 1.00000000000000000000000   * 2^26
```

Floats: Guidelines

```
float a = .2f;
for(int i = 0; i < 20; i++){
    a = 6*a - 1;
    std::cout << a << std::endl;
}
```

```
0.2
0.2
0.200002
0.20001
0.200062
0.200371
0.202225
0.213348
0.28009
0.680542
3.08325
```

«**Vermeidet** die Subtraktion von Zahlen mit **ähnlicher Grösse**»

(Rundungsfehler)

Funktionen

Schreibe ein Programm, welches für drei Zaheln a, b und c die Summe a! + b! + c! zurückgibt.

- Wir müssen drei mal den selben Code kopieren um die Fakultät zu berechnen

```
unsigned int a,b,c;
std::cin >> a >> b >> c;

unsigned int a_fact = 1;
for(int i = 1; i <= a; i++){
        a_fact *= i;
}

unsigned int b_fact = 1;
for(int i = 1; i <= b; i++){
        b_fact *= i;
}

unsigned int c_fact = 1;
for(int i = 1; i <= c; i++){
        c_fact *= i;
}

std::cout << a_fact + b_fact + c_fact << std::endl;
```

Funktionen

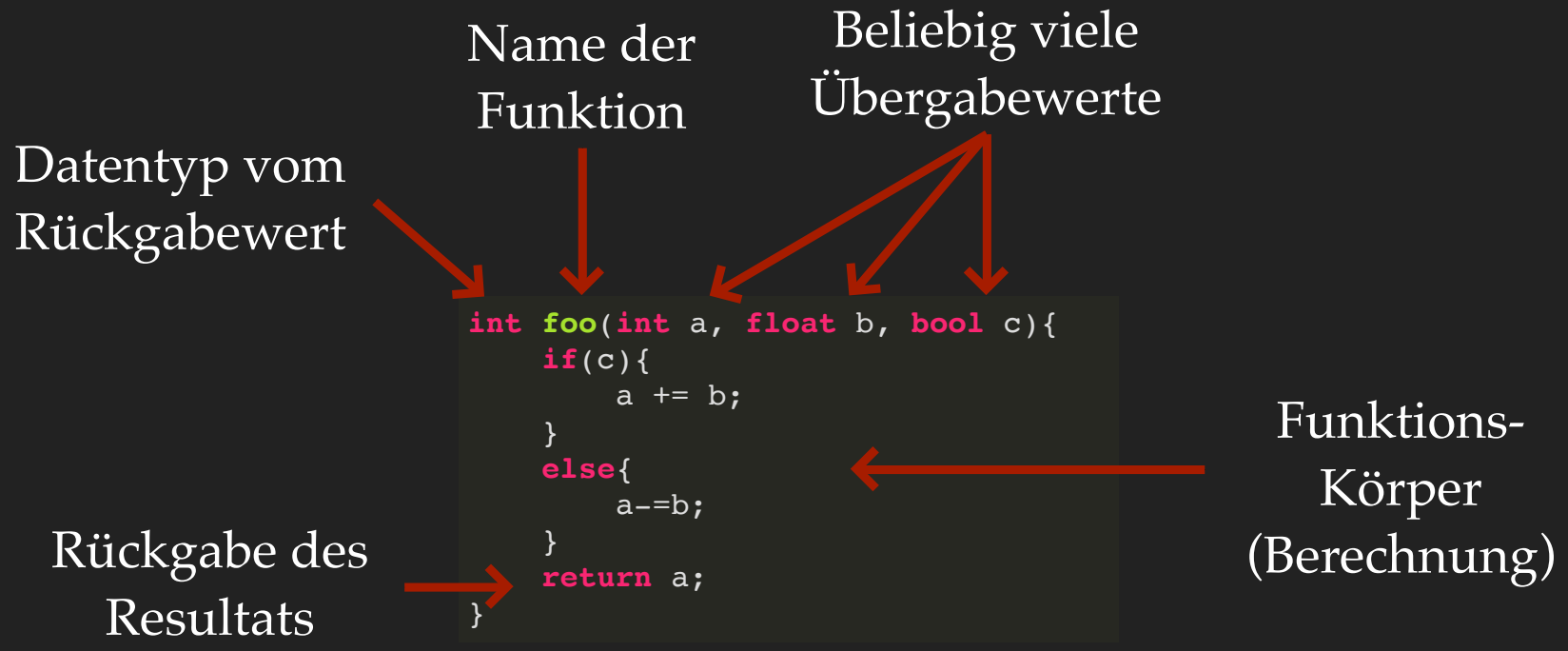
Wie können wir verhindern, immer wieder den selben Code zu kopieren?

- Wir können die Fakultät in eine "Funktion" packen, so dass wir den Algorithmus nur einmal schreiben müssen.

```
#include <iostream>

unsigned int fact(unsigned int n){
    unsigned int out = 1;
    for(int i = 0; i < n; i++){
        out *= i;
    }
    return out;
}


int main(){
    unsigned int a,b,c;
    std::cin >> a >> b >> c;

    std::cout << fact(a) + fact(b) + fact(c) << std::
    return 0;
}
```

Funktionen

Eine Funktion hat die folgenden Komponenten:



Funktionen

Findet die 10 Fehler!

```
bool isPrime(unsigned int n){
    for(unsigned int i = 2; i <= n/2; i++){
        if(n % i == 0){
                return false;
        }
    }
    return true;
}
```

Funktionen

```
//PRE:  left <= right
//POST: returns true iff x is in the Interval [left, right]
bool inInterval(double x, double left, double right){
        return x >= left && x <= right;
}
```

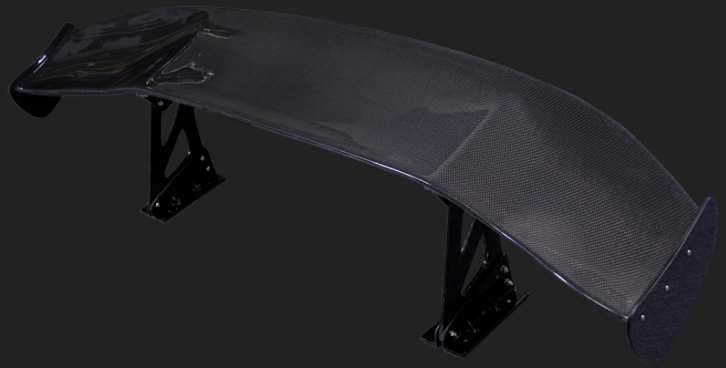
Precondition:

- Was muss bei Funktionsaufruf gelten?
- Spezifiziert Definitionsbereich der Funktion.

Postcondition:

- Was gilt nach Funktionsaufruf?
- Spezifiziert Wert und Effekt des Funktionsaufrufes.

Vorbesprechung



Pascal Schärli *15.03.2019*

ETH zürich

1: Float Representation

Gegeben ein binäres Float-System:

- $\beta = 2$ (binär)
- $p = 4$ (präzision)
- $e \in [-3; 3]$ (Exponent range)

1. Schreibe das System in der Notation $F^*(b, p, e_{min}, e_{max})$
2. Stelle 3.1416_{10}, 2.718_{10}, 7_{10} und 0.11_{10} dar in diesem System
3. Wandelt eurer Resultat von 2. wieder in Dezimal und bestimmt den Fehler
4. Berechnet $2.718_{10}+3.1416_{10}+0.11_{10}$ in diesem System

Tipp:
Normalisiert $\rightarrow$ Zahl vor Komma darf nicht 0 sein!

2: Point on Parabola ?

Schreibt ein Programm, welches bestimmt ob ein Punkt (x, y) auf der Parabel $g(x) = 0.9 \cdot x^2 + 1.3 \cdot x - 0.7$ liegt

Tipps:

- Direkter Vergleich wird nicht funktionieren (Guidlines)
- Baut eine Fehlertoleranz ein

$$a == b \rightarrow |a - b| < \epsilon$$

3: Rounding

Schreibt eine *Funktion*, welche einen double auf den nächsten Integer runden kann.

Tipp:

- Serie 1 Aufgabe 3

4: Binary Expansion

Schreibt ein Programm, welches ein x mit $0 \leq x \leq 2$ als Binärzahl im folgenden Format schreiben kann:
$b_0.b_1b_2b_3...b_{15}$

Tipp:
Wendet diesen Algorithmus an:

Algorithmus:
$b_i \rightarrow$ Binärziffer
$x_i \rightarrow$ Dezimalzahl
Folgende Schritte wiederholen:

1. b_i : Zahl vor Komma
2. $x_{i+1} = 2 * (x_i - b_i)$

5: Fixing Functions

```
double invert (double x) {
    double result;
    if (x != 0)
        result = 1 / x;
    return result;
}
```

```
bool is_even (unsigned int i) {
    if (i % 2 == 0) return true;
}
```

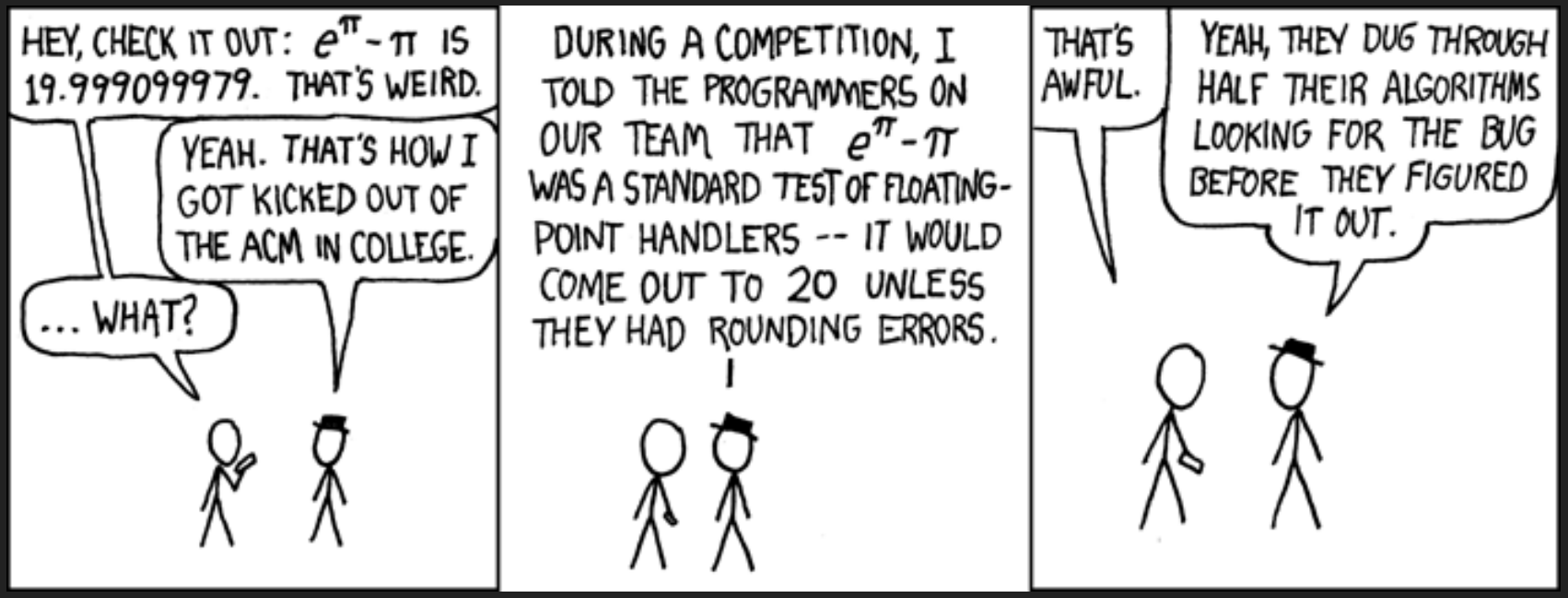
Behebe alle Probleme welche diese Funktionen haben und füge passende pre- und post conditions.

Tipp:

- Spielt selbst ein paar Eingabewerte durch und schaut was zurückkommt.

Program of the Week

Viel Spass!


HEY, CHECK IT OUT: $e^{\pi} - \pi$ IS 19.999099979. THAT'S WEIRD.
YEAH. THAT'S HOW I GOT KICKED OUT OF THE ACM IN COLLEGE.
... WHAT?
DURING A COMPETITION, I TOLD THE PROGRAMMERS ON OUR TEAM THAT $e^{\pi} - \pi$ WAS A STANDARD TEST OF FLOATING-POINT HANDLERS -- IT WOULD COME OUT TO 20 UNLESS THEY HAD ROUNDING ERRORS.
THAT'S AWFUL.
YEAH, THEY DUG THROUGH HALF THEIR ALGORITHMS LOOKING FOR THE BUG BEFORE THEY FIGURED IT OUT.