Funktionale Programmierung,Winter 2021/2022

1. Übungszettel
Abgabe: 29.10.2021
Katharina Klost
Seminar am PC 13 - Florian Alex
Arbeitsgruppe: Simeon Georgiev Vasilev und Tomás Proaño

---

2. Haskell als Taschenrechner

(a) `2 ^ 1023`

Ergebnis:

```
898846567431157953864652595394512366808988489471153286367150405788663379027504815663542386612037680105600569399356966788293948844072083112464237153197370621888839467124327426381511098006230470597265414760425028844190753411712314407369565552704136185816752553422931491199736229692398581524176781648121120686080
:: Num a => a
```

(b) `2 ** 1023`

Ergebnis:

```
8.98846567431158e307
:: Floating a => a
```

(c) `div 17 5`

```
3
:: Integral a => a
```

(d) `23 /= 23`

Ergebnis:

```
False
:: Bool
```

(e) `23 /= "23"`

Ergebnis:

```
<interactive>:5:1: error:
     • No instance for (Num [Char]) arising from the literal
     ‘23’
    • In the first argument of ‘(/=)’, namely ‘23’
      In the expression: 23 /= "23"
      In an equation for ‘it’: it = 23 /= "23"
```

(f) `sqrt (-1)`

Ergebnis:

```
NaN
:: Floating a => a
```

Grund: Die Quadratwurzel einer negativen Zahl existiert nicht in der Menge der reellen Zahlen.

(g) `23 < "23"`

Ergebnis:

```
<interactive>:7:1: error:
        • No instance for (Num [Char]) arising from the literal
    ‘23’
    • In the first argument of ‘(<)’, namely ‘23’
      In the expression: 23 < "23"
      In an equation for ‘it’: it = 23 < "23"
```

(h) `'2'<'a'`

Ergebnis:

```
True
:: Bool
```

(i) `exp 1`

Ergebnis:

```
2.718281828459045
:: Floating a => a
```

(j) `(cos 45) ^ 2 +(sin 45) ^ 2`

Ergebnis:

```
1.0
:: Floating a => a
```

(√)

die Begründungen fehlen bei fast allen Antworten.

5,5/10

3. Ganzzahlige Division (10 Punkte)

(a)

Laut der Dokumentation:

```
quot :: a -> a -> a
integer division truncated toward zero

div :: a -> a -> a infixl 7
integer division truncated toward negative infinity
```

Antwort: Mit den Methoden quot und div kann man ganze Zahlen dividieren. Der Unterschied ist, dass bei quot in Richtung 0 abgerundet wird und, dass bei div in Richtung negativer Unendlichkeit abgerundet wird.

√

(b)

immer?

Antwort: Bei positiven Zahlen kommt das gleiche Ergebnis aus und bei negativen Zahlen kommen unterschiedliche Ergebnisse aus.

(√)

(c)

Laut der Dokumentation:

```
integer modulus, satisfying
(x `div` y)*y + (x `mod` y) == x
```

Antwort: Mod gibt den Modulus der beiden Zahlen zurück. Dies ähnelt dem Rest, hat jedoch andere Regeln, wenn div eine negative Zahl zurückgibt.

welche?

Zum Beispiel:

- **div 10 5**

Ergebnis:
```
2
```

- **div (-10) 5**

Ergebnis:
```
<interactive>:12:1: error:
     • Non type-variable argument in the constraint: Num (a -> a
     -> a)
       (Use FlexibleContexts to permit this)
     • When checking the inferred type
         it :: forall a t.
     (Integral a, Num t, Num (a -> a -> a), Num (t -> a -> a ->
     a)) => a -> a -> a
```

- **div 10 (-5)**

Ergebnis:
```
<interactive>:14:1: error:
     • Non type-variable argument in the constraint: Num (a ->
     a)
       (Use FlexibleContexts to permit this)
     • When checking the inferred type
         it :: forall a. (Integral a, Num (a -> a)) => a -> a
```

- **div (-10) (-5)**

Ergebnis:
```
<interactive>:15:1: error:
     • Non type-variable argument in the constraint: Num (a -> a
     -> a)
       (Use FlexibleContexts to permit this)
     • When checking the inferred type
     it :: forall a. (Integral a, Num (a -> a -> a)) => a -> a -
     > a
```

- **mod 10 5**

Ergebnis:
```
0
```

- **mod (-10) 5**

Ergebnis:
```
<interactive>:17:1: error:
     • Non type-variable argument in the constraint: Num (a -> a
     -> a)
```

```
      (Use FlexibleContexts to permit this)
    • When checking the inferred type
        it :: forall a t.
      (Integral a, Num t, Num (a -> a -> a), Num (t -> a -> a ->
      a)) => a -> a -> a
```

- **mod 10 (-5)**

Ergebnis:

```
<interactive>:18:1: error:
      • Non type-variable argument in the constraint: Num (a ->
      a)
        (Use FlexibleContexts to permit this)
    • When checking the inferred type
        it :: forall a. (Integral a, Num (a -> a)) => a -> a
```

- **mod (-10) (-5)**

Ergebnis:

```
<interactive>:19:1: error:
      • Non type-variable argument in the constraint: Num (a -> a
      -> a)
        (Use FlexibleContexts to permit this)
    • When checking the inferred type
      it :: forall a. (Integral a, Num (a -> a -> a)) => a -> a -
      > a
```

Was bedeuten diese Fehler? Wie beantworten diese die Frage?

(d)

Antwort:

- rem:

Typsignatur: Integral a => a -> a -> a
Division von ganzen Zahlen, es gibt den Rest der Division der Argumente zurück.

- quot:

Typsignatur: Integral a => a -> a -> a
Division von ganzen Zahlen, es dividiert das erste Argument durch das zweite und verwirft den Rest.

Beispiele:

- -6 `rem` -2

Ergebnis:

```
<interactive>:24:1: error:
    Precedence parsing error
```

```
    cannot mix ‘mod’ [infixl 7] and prefix `-' [infixl 6] in
    the same infix expression
```

- -6 `rem` 2

Ergebnis:

```
0
```

- 6 `rem` -2

Ergebnis:

```
<interactive>:25:1: error:
    Precedence parsing error
      cannot mix ‘mod’ [infixl 7] and prefix `-' [infixl 6] in
      the same infix expression
```

- 6 `rem` 2

Ergebnis:

```
    0
```

- -6 `quot` -2

Ergebnis:

```
<interactive>:27:1: error:
    Precedence parsing error
      cannot mix ‘quot’ [infixl 7] and prefix `-' [infixl 6] in
      the same infix expression
```

- -6 `quot` 2

Ergebnis:

```
-3
```

- 6 `quot` -2

Ergebnis:

```
<interactive>:29:1: error:
    Precedence parsing error
    cannot mix ‘quot’ [infixl 7] and prefix `-' [infixl 6] in
    the same infix expression
```

- 6 `quot` 2

Beobachtung: mod und div sind nicht gleich, wenn das zweite Argument negativ ist:

2 `mod` (-3) == -1

2 `rem` (-3) == 2

(e)

mod: Division von ganzen Zahlen, es gibt den Rest der Division der Argumente zurück.

div: Es gibt zurück, wie oft die erste Zahl durch die zweite geteilt werden kann.

quot: Division von ganzen Zahlen, es dividiert das erste Argument durch das zweite und verwirft den Rest.

rem: Division von ganzen Zahlen, es gibt den Rest der Division der Argumente zurück. Der Unterschied mit mod ist, dass rem schneller ist. mod und rem verhalten sich anders, wenn das zweite Argument negativ ist. Wieso?

(✓)

5/10

4. Windchill-Temperatur (4 Punkte)

(a) Funktionssignatur: t_{chill} : : Int -> Int -> Int -> Int -> Int-> Int ✗ Wieso so viele Eingaben? Die Eingaben sind nur t und v + t und v sind vom Typ Double

(b) windchill = 13.12+0.6215 * t + (0.3965 * t - 11.37) * (v**0.16)

die Funktionen unterscheiden sich von denen aus dem Quellcode (ich habe diese hier bewertet)

1/4

5. Zinsen (Unterfunktionen)(6 Punkte)

(a)

endwert : : double->double->double
endwert kapital zinssatz = kapital+(zinsen kapital zinssatz) ✓

(b)

endwert2 : : double->double->double
endwert2 kapital2 zinsfuss2 = endwert (endwert kapital2 zinsfuss2) zinsfuss2 ✓

(c) Ja.

Wieso?

5/6

76.5/30