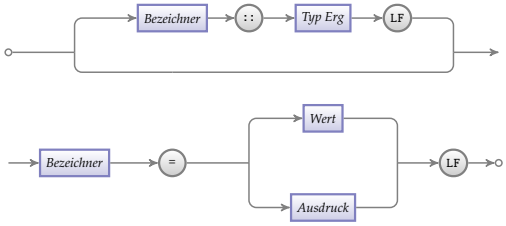
1 Kommentare

- Zeilenkommentar: `-- Text`
- Blockkommentar: `{- Text -}`

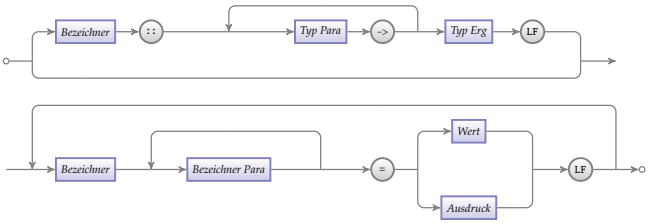
2 Konstanten

```
pi :: Double
pi = 3.14
```



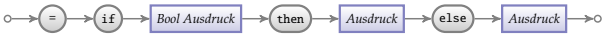
3 Funktionen

```
plus :: Int -> Int -> Int
plus a b = a + b
```



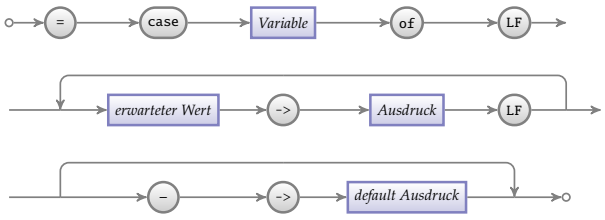
4 If-Then-Else

```
ggT a b = if b == 0
          then a
          else ggT b (mod a b)
```



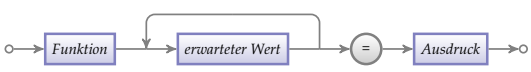
5 Case Of

```
ggT a b = case b of
          0 -> a
          _ -> ggT b (mod a b)
```



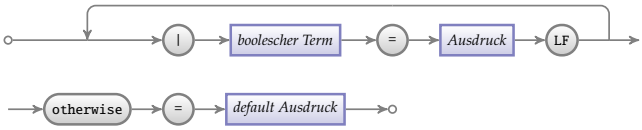
6 Pattern Matching

```
ggT a 0 = a
ggT a b = ggT b (mod a b)
```



7 Guards

```
ggT a b | b == 0     = a
        | otherwise =
            ggT b (mod a b)
```



8 Module

```
module Wurf(weite, square) where
weite :: Double -> Double -> Double
weite v0 phi = ((square v0) / 9.81) * sin (2 * phi)
square :: Double -> Double
square x = x * x

module Foo where
import Wurf hiding (weite)
bar ... = ... (square a) ...
```

9 Eigene Datentypen

```
data Point = Point{x :: Double, y :: Double}

data Shape = Circle{center :: Point,
    radius :: Double}
  | Rectangle{point :: Point, width :: Double,
    height :: Double}
  | Triangle{point1 :: Point, point2 :: Point,
    point3 :: Point}
```

10 Typparameter

Keine Typparameter in Datentyp-Definitionen!

```
data (Eq a, Ord a) => Pair a b =
  PairConst {first :: a, second :: b} deriving(Show)
```

Besser:

```
data Pair a b = PairConst a b deriving(Show)

instance Eq a => Eq (Pair a b) where
  (PairConst i1 _) == (PairConst i2 _) = i1 == i2

instance Ord a => Ord (Pair a b) where
   (PairConst i1 _) <= (PairConst i2 _) = i1 <= i2

bubbleSort :: (Ord a) => [(Pair a b)] -> [(Pair a b)]
bubbleSort [] = []
bubbleSort (x:xs) = step $ foldl go (x,[]) xs where
  go (y,acc) x = (min x y, max x y : acc)
  step (x,acc) = x : bubbleSort acc
```

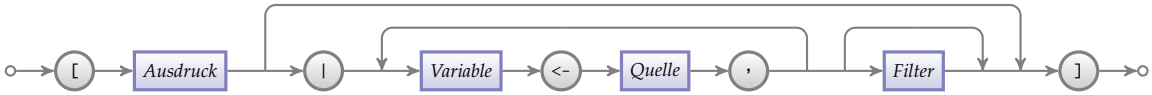
$ ist nichts anderes als:

```
(7 + 6) / (5 + 3) = (7 + 6) / $ 5 + 3
```

Die „Klammerung“ geht bis zum Zeilenende.

11 Listen

```
primes = sieve [2..]
  where
    sieves (p:xs) = p:sieves [x|x<- xs, mod x p > 0]
```



12 Operationen

Bezeichner	Typ	Bedeutung
`(+)`	`a -> a -> a`	Addition
`(-)`	`a -> a -> a`	Subtraktion
`(*)`	`a -> a -> a`	Multiplikation
`negate`	`a -> a`	Negation
`abs`	`a -> a`	Absolutbetrag
`signum`	`a -> a`	Vorzeichenbildung

Tabelle 1: Für die Typen `Int`, `Integer`, `Float` und `Double`

Bezeichner	Typ	Bedeutung
`succ`	`a -> a`	Nachfolgerbildung
`pred`	`a -> a`	Vorgängerbildung
`div`	`a -> a -> a`	ganzzahlige Division, Ergebnis wird abgerundet
`mod`	`a -> a -> a`	zur ganzzahligen Division `div` gehörender Rest
`quot`	`a -> a -> a`	ganzzahlige Division, Ergebnis wird Richtung 0 gerundet
`rem`	`a -> a -> a`	zur ganzzahligen Division `quot` gehörender Rest

Tabelle 2: Für die Typen `Int` und `Integer`

Bezeichner	Typ	Bedeutung
`(/)`	`a -> a -> a`	Division
`recip`	`a -> a`	Kehrwertbildung
`(**)`	`a -> a -> a`	Potenzieren
`sqrt`	`a -> a`	Ziehen der Quadratwurzel

Tabelle 3: Für die Typen `Float` und `Double`

Bezeichner	Typ	Bedeutung
`(||)`	`a -> a -> a`	Oder
`(&&)`	`a -> a -> a`	Und
`not`	`a -> a`	Verneinung

Tabelle 4: Für den Typ `Bool`