Geomeetriliste kujundite ümbermõõt

- Kujundite ümbermõõdu leidmise funktsioonil on neli võrdust (1 iga konstruktori kohta).
- Variandid `Rectangle` ja `RtTriangle` jaoks on lihtsad:

```
perimeter :: Shape -> Float
perimeter (Rectangle  s1 s2) = 2*(s1+s2)
perimeter (RtTriangle s1 s2)
             = s1 + s2 + sqrt(s1^2+s2^2)
```

- `Polygon` on samuti lihtne, kui oskame külgede pikkusi arvutada:

```
perimeter (Polygon vs) = foldl (+) 0 (sides vs)
```


Geomeetriliste kujundite ümbermõõt

- Külgede pikkuste leidmine rekursiivselt:

```
sides        :: [Vertex] -> [Side]
sides  []    = []
sides (v:vs) = aux v vs
   where aux v1 (v2:vs')
                 = distBetween v1 v2 : aux v2 vs'
         aux vn [] = [distBetween vn v]
```

Geomeetriliste kujundite ümbermõõt

- Funktsioon `zipWith`

```
zipWith :: (a->b->c) -> [a]->[b]->[c]
zipWith f (a:as) (b:bs)
              =  f a b : zipWith f as bs
zipWith _ _ _ =  []
```

- Külgede pikkuse leidmine mitterekursiivselt:

```
sides vs = zipWith distBetween vs
                         (tail vs ++ [head vs])
```


Geomeetriliste kujundite ümbermõõt

- Ellipsi ümbermõõt on defineeritud lõpmatu summana; kui r_1 ja r_2 on ellipsi poolteljed $(r_1 \geq r_2)$, siis:

$$p = 2\pi r_1 (1 - \sum_{i=0}^{\infty} s_i),$$

kus

$$\begin{aligned} e &= \frac{\sqrt{r_1^2 - r_2^2}}{r_1} \\ s_1 &= \frac{e^2}{4} \\ s_{i+1} &= \frac{s_i(2i-1)(2i-3)e^2}{4i^2} \end{aligned}$$

Kõrgemat-järku funktsioonid

- Funktsioon `scanl`

$$\texttt{scanl}\ (\oplus)\ e\ [x_1, x_2, x_3]$$
$$\Longrightarrow [e, e \oplus x1, (e \oplus x_1) \oplus x_2, ((e \oplus x_1) \oplus x_2) \oplus x_3]$$

- Definitsioon:

```
scanl :: (a -> b -> a) -> a -> [b] -> [a]
scanl f q  []    = [q]
scanl f q (x:xs) = q : scanl f (f q x) xs
```


Geomeetriliste kujundite ümbermõõt

- Jada järgmise elemendi leidmine:

```
nextEl      :: Float -> Float -> Float -> Float
nextEl e s i = s*(2*i-1)*(2*i-3)*(e^2) / (4*i^2)
```

- Nüüd on jada esitatav `scanl` abil

```
s = scanl (nextEl e) (0.25*e^2) [2..]
```

- Näiteks, kui `r1 = 2.1` ja `r2 = 1.5`, siis `e = 0.699854` ning

```
s = [0.122449, 0.0112453, 0.00229496,
     0.000614721, 0.000189685, 6.38736e-05, ...
```

Geomeetriliste kujundite ümbermõõt

- Funktsioon `takeWhile`

```
takeWhile :: (a -> Bool) -> [a] -> [a]
takeWhile p []          = []
takeWhile p (x:xs) | p x = x : takeWhile p xs
                   | otherwise = []
```

- Jada summa leidmiseks võtame jadast soovitud täpsuseni lõpliku prefiksi

```
epsilon = 0.0001 :: Float

sSum = foldl (+) 0 (takeWhile (> epsilon) s)
```


Geomeetriliste kujundite ümbermõõt

- Ellipsi pindala:

```
perimeter (Ellipse r1 r2)
      | r1 > r2   = ellipsePerim r1 r2
      | otherwise = ellipsePerim r2 r1
  where ellipsePerim r1 r2
        = let e  = sqrt (r1^2 - r2^2) / r1
              s0 = 0.25*e^2
              s  = scanl (nextEl e) s0 [2..]
              test x = x > epsilon
              sSum = sum (takeWhile test s)
          in 2*r1*pi*(1-sSum)
```

Puud

- Puud on arvutiteaduses ja programmeerimises üks olulisemaid andmestruktuure
- On piiramatu suurusega, kuid tavaliselt lõplikud
  - laiskades keeltes võivad olla ka lõpmatud

```
data SimpleTree = Leaf
                | Branch SimpleTree SimpleTree
```

- Tippudes võivad olla mingit teist tüüpi väärtused:

```
data IntegerTree = Leaf Integer
                 | Branch IntegerTree IntegerTree
```


Puud

- Tippudes olevad väärtused võivad olla polümorfsed:

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
data InternalTree a = ILeaf
                    | IBranch a (InternalTree a)
                                (InternalTree a)
```

- Lehed ja vahetipud võivad olla erinevat tüüpi:

```
data FancyTree a b = FLeaf a
                   | FBranch b (FancyTree a b)
                               (FancyTree a b)
```

Puud

- Puud võivad erinevalt hargneda:
  - lineaarsed puud (= listid):

    ```
    data List  a = Nil | MkList a (List a)
    ```

  - suvaliselt hargnevad puud:

    ```
    data GTree a = Node a [GTree a]
    ```

  - suvaliselt hargnevad "lehtpuud":

    ```
    data PTree a = PLeaf a
                 | PBranch (Int -> PTree a)
    ```


Funktsioone puudel

- Puu elementide teisendamine:

```
mapTree :: (a->b) -> Tree a -> Tree b
mapTree f (Leaf x)       = Leaf (f x)
mapTree f (Branch t1 t2) = Branch (mapTree f t1)
                                  (mapTree f t2)
```

- Puu elementide "kokku korjamine":

```
fringe :: Tree a -> [a]
fringe (Leaf x)       = [x]
fringe (Branch t1 t2) = fringe t1 ++ fringe t2
```

Funktsioone puudel

- Puu suuruse leidmine:

```
treeSize :: Tree a -> Integer
treeSize (Leaf x)       = 1
treeSize (Branch t1 t2) = treeSize t1
                        + treeSize t2
```

- Puu kõrguse leidmine:

```
treeHeight :: Tree a -> Integer
treeHeight (Leaf x)       = 0
treeHeight (Branch t1 t2) = 1
                          + max (treeHeight t1)
                                (treeHeight t2)
```


Funktsioone puudel

- Puude `fold`:

```
foldTree :: (b -> b -> b) -> (a -> b)
                          -> Tree a -> b
foldTree b l (Leaf x)       = l x
foldTree b l (Branch t1 t2) = b (foldTree b l t1)
                                (foldTree b l t2)
```

- Näiteid:

```
mapTree f  = foldTree Branch (Leaf . f)
fringe     = foldTree (++) (:[])
treeSize   = foldTree (+) (const 1)
treeHeight = foldTree (((1+).) . max) (const 0)
```

Aritmeetilised avaldised

- Aritmeetilised avaldised puuna:

```
data ArithOp = Add | Sub | Mul | Div
type Expr    = FancyTree Float ArithOp
```

- Otsene definitsioon:

```
data Expr = C Float | Add Expr Expr | Sub Expr Expr
                    | Mul Expr Expr | Div Expr Expr
```

- Definitsioon kasutades infix-konstruktoreid:

```
data Expr = C Float | Expr :+ Expr | Expr :- Expr
                    | Expr :* Expr | Expr :/ Expr
```


Aritmeetilised avaldised

- Aritmeetiliste avaldiste väärtustamine:

```
evaluate :: Expr -> Float
evaluate (C x) = x
evaluate (e1 :+ e2) = evaluate e1 + evaluate e2
evaluate (e1 :- e2) = evaluate e1 - evaluate e2
evaluate (e1 :* e2) = evaluate e1 * evaluate e2
evaluate (e1 :/ e2) = evaluate e1 / evaluate e2
```

- Näide:

```
e1 = (C 10 :+ (C 8 :/ C 2)) :* (C 7 :- C 4)
evaluate e1   ===>  42.0
```

Lõpmatud puud

- Kõigi naturaalarvude summa:

```
sumFromN n = C n :+ (sumFromN (n+1))
sumAll     = sumFromN 1

add1 (C x) = C (x+1)
add1 (e1 :+ e2) = add1 e1 :+ add1 e2
add1 (e1 :- e2) = add1 e1 :- add1 e2
add1 (e1 :* e2) = add1 e1 :* add1 e2
add1 (e1 :/ e2) = add1 e1 :/ add1 e2

sumAll2 = C 1 :+ add1 sumAll
```


Lõpmatud puud

- Lõpmatu summa visualiseerimine:

```
showE 0 _          = "..."
showE n (C x)      = show x
showE n (e1 :+ e2) = "(" ++ showE (n-1) e1 ++
                     "+" ++ showE (n-1) e2 ++ ")"

Main> showE 5 sumAll
"(1.0+(2.0+(3.0+(4.0+(...+...)))))"
Main> showE 5 sumAll2
"(1.0+(2.0+(3.0+(4.0+(...+...)))))"
```