Ülesanne 1

Vektorid defineeritud kui:

```
data Vect : (k : Nat) → (a : Type) → Type where
  Nil  : Vect Z a
  (::) : a → Vect k a → Vect (S k) a

test : Vect 3 Int
test = [2,3,4]
```

Defineerige vektori pea leidmise funktsioon! Näiteks:

```
Main> head test
2
```

Ülesanne 2

Vektorid defineeritud kui:

```
data Vect : (k : Nat) → (a : Type) → Type where
  Nil  : Vect Z a
  (::) : a → Vect k a → Vect (S k) a

test : Vect 3 Int
test = [2,3,4]
```

Defineerige vektori *saba* leidmise funktsioon! Näiteks:

```
Main> tail test
[3, 4]
```

Ülesanne 3

Vektorid defineeritud kui:

```
data Vect : (k : Nat) → (a : Type) → Type where
  Nil  : Vect Z a
  (::) : a → Vect k a → Vect (S k) a

test : Vect 3 Int
test = [2,3,4]
```

Defineerige vektori elementide summa funktsioon! Näiteks:

```
Main> sumVect test
9
```

Ülesanne 4

Vektorid defineeritud kui:

```
data Vect : (k : Nat) → (a : Type) → Type where
  Nil  : Vect Z a
  (::) : a → Vect k a → Vect (S k) a

appendVect : Vect n a → Vect m a → Vect (n + m) a
appendVect []        ys = ys
appendVect (x::xs) ys = x :: appendVect xs ys

test : Vect 3 Int
test = [2,3,4]
```

Defineerige vektori ümberpööramine! Näiteks:

```
Main> reverseVect test
[4,3,2]
```

Ülesanne 5

Vektorid defineeritud kui:

```
data Vect : (k : Nat) → (a : Type) → Type where
  Nil  : Vect Z a
  (::) : a → Vect k a → Vect (S k) a

myFilter : -- tüüp --
myFilter p [] = []
myFilter p (x :: xs) with (p x)
    _ | True = x :: myFilter p xs
    _ | False = myFilter p xs
```

Mis on myFilter tüüp, et töötaks näiteks:

```
Main> myFilter (λ x ⇒ x `mod`2 == 0) [2,3,4,5,6,7]
[2, 4, 6]
```

Ülesanne 6

```
data TreeShape : Type where
  LeafShape : TreeShape
  NodeShape : (l : TreeShape) → (r : TreeShape) → TreeShape

data Tree : TreeShape → Type → Type where
  Leaf : Tree LeafShape a
  Node : (left : Tree l a) → (this : a) → (right : Tree r a) →
       Tree (NodeShape l r) a

tr1 : Tree (NodeShape LeafShape (NodeShape LeafShape LeafShape)) Int
tr1 = Node Leaf 1 (Node Leaf 2 Leaf)

tr2 : Tree (NodeShape LeafShape (NodeShape LeafShape LeafShape)) Int
tr2 = Node Leaf 3 (Node Leaf 1 Leaf)

addTree : Num a ⇒ Tree s a → Tree s a → Tree s a
addTree = ?add_tree_rhs
```

Implementeeri puude liitmine, et töötaks näiteks:

```
Main> addTree tr1 tr1
Node Leaf 2 (Node Leaf 4 Leaf)
Main> addTree tr1 tr2
Node Leaf 4 (Node Leaf 3 Leaf)
```

Ülesanne 7

```
data TreeShape : Type where
  LeafShape : TreeShape
  NodeShape : (l : TreeShape) → (r : TreeShape) → TreeShape

data Tree : TreeShape → Type → Type where
  Leaf : Tree LeafShape a
  Node : (left : Tree l a) → (this : a) → (right : Tree r a) →
       Tree (NodeShape l r) a

tr1 : Tree (NodeShape LeafShape (NodeShape LeafShape LeafShape)) Int
tr1 = Node Leaf 1 (Node Leaf 2 Leaf)

tr2 : Tree (NodeShape LeafShape (NodeShape LeafShape LeafShape)) Int
tr2 = Node Leaf 3 (Node Leaf 1 Leaf)
```

Implementeeri puude vektoriks tegemine, et töötaks näiteks:

```
Main> treeToVec tr1
[1, 2]
Main> treeToVec tr2
[3, 1]
```

Probleem

Tahame listi indekseerida funktsiooniga: `Nat → List a → a`

Probleem:

- List ei pruugi sisaldada `n`-ndat elementi!
- List võib olla täiesti tühi! Siis ei saa isegi valet elementi tagastada.

Võime teha nii: `Nat → List a → Maybe a`.

Töötab, kuid pole optimaalne juhul, kui listis on piisavalt elemente.

Listi tüüp võiks sisaldada infot tema pikkuse kohta.

Pikkusega listid

- Tavalised listid (uue süntaksi abil)

```
data List : Type → Type where
    Nil : List a
    (::) : a → List a → List a
```

- Paneme pikkused külge

```
data Vec : Nat → Type → Type where
    Nil : Vec 0 a
    (::) : a → Vec n a → Vec (1+n) a

test5 : Vec 3 Int
test5 = [1,2,3]
```

- selekteerimine

```
nth : (n:Nat) → Vec (1+n+m) a → a
nth 0     (x :: xs) = x
nth (S k) (x :: xs) = nth k xs
```

- konkateneerimine

```
concatVec : Vec n a → Vec m a → Vec (n+m) a
concatVec []       ys = ys
concatVec (x :: y) ys = x :: concatVec y ys
```

- Vektorite map

```
Functor (Vec n) where
 -- map : (a → b) → Vec n a → Vec n b
    map f []       = []
    map f (x :: y) = f x :: map f y
```

NB! Definitsioon sama mis listidel!

Funktsiooni `nth n` tüüp `Vec (1+n+m) a → a` sõltub `n`-i väärtusest.

- `nth 0 : Vec (1+m) a → a`
- `nth 2 : Vec (3+m) a → a`

Sõltuvad tüübid võimaldavad arvutamist ja selle tõestamist koos teha.