

# Gequoteerde zitting Haskell: Phonebooks

NAAM:

RICHTING:

## Enkele praktische afspraken

- Je krijgt twee uur om deze opdracht **individueel** op te lossen.
- Je raadpleegt enkel de Haskell slides (eventueel in afgedrukte vorm met handgeschreven nota's), de cursustekst en de oefeningen die via Toledo voor dit vak ter beschikking gesteld zijn. Je mag ook de manuals vermeld op Toledo en eventueel Hoogle raadplegen.
- In de map **1617\_Gequoteerde/Haskell\_Woensdag** op Toledo vind je de bestanden `Phonebook.hs` en `PhonebookTest.hs`. Ook de `indien` module staat daar.
  - **Download en open** het bestand `Phonebook.hs`, hierin staat reeds een template voor je oplossing.
  - Als eerste vul je bovenaan je naam, studentnummer en richting in.

```
-- Jan Jansen
-- r0123456
-- master cw
```
  - Je mag **extra functies en types importeren!**
  - Voor elke opdracht zijn een aantal functies reeds gedefinieerd met een bijbehorende typesignatuur. Deze typesignatuur mag niet gewijzigd worden. Vervang telkens `undefined` met jouw implementatie. Je mag argumenten voor het gelijkheidsteken zetten of weghalen. Het is natuurlijk ook altijd toegestaan om extra (hulp)functies te schrijven.
  - Je kan je oplossing testen m.b.v. `PhonebookTest.hs`. Dit doe je door in de map waarin de twee `.hs` bestanden staan het volgende commando uit te voeren:

```
runhaskell PhonebookTest.hs
```

**N.B.** dat alle testen slagen betekent niet per se dat je programma helemaal correct is of dat je het maximum van de punten verdient.
  - Na twee uur of wanneer je klaar bent, dien je het bestand `Phonebook.hs` in via Toledo.

# Telefoonboeken

Let erop dat u de lijnen deriving (Eq,Show) niet verwijdert wanneer u de data types implementeert.

Het niet toegelaten om type signatures te veranderen!

Het *correct* gebruik van hogere-orde functies kan een positief effect hebben op uw score.

## 1 Entry

Een **Entry** is de basiseenheid in een telefoonboek. In deze oefening bevat een **Entry** een naam en een telefoonnummer.

### Opgave 1

Vervolledig de definities van **Entry**, **mkEntry**, **name** en **phone**. Een **Entry** bevat een veld van het type **Name** en een veld van het type **PhoneNumber**. De functies **name** en **phone** beelden een **Entry** af op het overeenkomstige veld.

De types zien er als volgt uit:

---

```
mkEntry :: Name -> PhoneNumber -> Entry
name    :: Entry -> Name
phone   :: Entry -> PhoneNumber
```

---

De types **Name** en **PhoneNumber**, die respectievelijk namen en telefoonnummers voorstellen, zijn reeds gedefiniëerd.

## 2 De Index-typeclass

In deze sectie wordt de typeclass **Index** gedefiniëerd. Deze typeclass vormt een abstractie over een collectie waarin **Entrys** kunnen worden opgezocht volgens een key. De klasse bevat 4 functies:

- **findEntry** :: Eq k => k -> i k -> Maybe Entry geeft Just e terug waarbij e een Entry uit i k is die met een gegeven key (van type k) overeenkomt; of Nothing als een dergelijke e niet bestaat.
- **empty** :: Eq k => i k maakt een i k die geen Entrys bevat.
- **singleton** :: Eq k => k -> Entry -> i k maakt een i k dat enkel de gegeven Entry bevat, geïndexeerd door de geven sleutel.

- `(<+>) :: Eq k => i k -> i k -> i k` combineert twee `i k`s. Het resultaat is de unie van de elementen van beide `i k`s. Als er conflicten zijn, dit wil zeggen, er zijn twee verschillende **Entrys** door dezelfde key worden aangeduid, geef dan de voorkeur aan de Entry uit de eerste `i k`. Met andere woorden, de volgende eigenschap moet gelden: `findEntry k (singleton k e1 <+> singleton k e2) == Just e1` voor alle `k`.

In deze opgave gebruiken we een eenvoudige variant van een data type dat aan deze interface voldoet: een associatielijst. Dit is een lijst die paren van een keys met **Entrys** bevat.

## Opgave 2

1. Vervolledig de definitie van **Assoc**, zodat het precies één veld bevat: een associatielijst die geparameteriseerd is in het type van de key die gebruikt wordt. Bijvoorbeeld:

---

```
> :t MkAssoc [(1,bill),(2,bob),(3,jeb)]
MkAssoc [(1,bill),(2,bob),(3,jeb)] :: Assoc Int

> :t MkAssoc [("a",bill),("b",bob),("c",jeb)]
MkAssoc [(1,bill),(2,bob),(3,jeb)] :: Assoc String

> :t MkAssoc [(1.0,bill),(2.0,bob),(3.0,jeb)]
MkAssoc [(1,bill),(2,bob),(3,jeb)] :: Assoc Double
```

---

2. Vul nu de **Index**-instance voor **Assoc** verder aan.

Voorbeelden:

---

```
> findEntry "Bill" (empty :: Assoc String)
Nothing

> findEntry "Bob" (singleton "Bob" bob :: Assoc String)
Just (MkEntry "Bob" [32,444,124])

> let sBob = singleton (name bob) bob
> let sJeb = singleton (name jeb) jeb
> findEntry "Jebediah" (sBob <+> sJeb :: Assoc String)
Just (MkEntry "Jebediah" [32,444,125])
```

---

## 3 Phonebook

Een **PhoneBook** is een verzameling van **Entrys**, die op twee verschillende manieren opgezocht kunnen worden: volgens naam en volgens telefoonnummer.

## Opgave 3

- Vervolledig de definitie van `PhoneBook`. Een telefoonboek bevat 3 velden: (1) de eigenaar van het telefoonboek (van het type `Entry`), (2) een index voor namen (van het type `Assoc Name`) en (3) een index voor telefoonnummers (`Assoc PhoneNumber`).
- Vervolledig eveneens de definities voor `names`, `phones` en `owner`. Deze functies beelden een `PhoneBook` af op het overeenkomstige veld en hebben de volgende types:

---

```
names    :: PhoneBook -> Assoc Name
phones   :: PhoneBook -> Assoc PhoneNumber
owner    :: PhoneBook -> Entry
```

---

## Opgave 4

Gebruik functies uit de vorige opgave en uit de `Index`-typeclass bij het schrijven van de functies uit deze opgave:

- Implementeerde functie `byName` die, geven een `Name` zoekt naar een overeenkomstige `Entry` in de `Assoc` die wordt teruggegeven door `names`. Op een gelijkaardige manier zoekt `byPhone` naar een `Entry` in `phones`.
- Implementeer de functie `emptyBook` die een lege telefoonboek maakt (die dus geen `Entrys` bevat), gegeven de `Entry` van de eigenaar. Dit impliceert dat `byName n emptyBook` en `byPhone p emptyBook` de waarde `Nothing` moeten teruggeven voor eender welke `n` en `p`.

Bijvoorbeeld:

---

```
> byName "Bill" (emptyBook bob)
Nothing

> byName "Bob" (emptyBook bob)
Nothing

> byPhone [32,444,124] (emptyBook bob)
Nothing
```

---

- Implementeer de functie `addToBook` die een `Entry` toevoegt aan een reeds bestaande `PhoneBook`. Het geval wanneer een het telefoonboek al een conflicterende `Entry` bevat is voor deze oefening niet relevant, en hier moet dus geen rekening mee worden gehouden.

**Hint:** gebruik `<+>` en `singleton`.

Voorbeeld:

---

```
> find "Bob" (addToBook bob (emptyBook bill))
MkEntry "Bob" [32,444,124]
```

---

- d. Implementeer de functie `fromEntries` die van een lijst met `Entrys` een `PhoneBook` maakt die al deze entries bevat.

**Hint:** gebruik de functies die u in deze sectie gedefiniëerd hebt.

Voorbeeld:

---

```
> let billsbook = fromEntries bill [bob,jeb]
> byName "Bob" billsbook
MkEntry "Bob" [32,444,124]
```

---

## 4 Caller ID

De meeste moderne telefoons hebben een schermje waarop het nummer van een binnenkomende telefoonoproep af te lezen valt. Als het telefoonnummer echter opgeslagen is in de interne telefoonboek van de telefoon, dan wordt in de plaats hiervan de naam van de beller getoond.

In deze opgave wordt een telefoon gemodelleerd door het `Telephone` datatype, dat voorgedefiniëerd is. Een telefoon heeft twee velden: het eerste veld, `number`, is het eigen nummer van de telefoon (van het type `PhoneNumber`). Het tweede veld, `receive`, is een *functie* van het type `PhoneNumber -> IO()`, die het gedrag van de telefoon beschrijft. Het argument van deze functie is het nummer van de beller, het resultaat is een `IO`-actie, bijvoorbeeld het afdrukken van het nummer van de beller.

### Opgave 5

Implementeer de functie `callerID :: PhoneBook -> Telephone` die een `Telephone` maakt waarvan het nummer overeenkomt met het nummer van de eigenaar van het telefoonboek. Het gedrag van de telefoon is als volgt: als het nummer van de beller in het telefoonboek staat, dan wordt de *naam* van de beller afgedrukt, anders wordt het *nummer* afgedrukt. Daarna print de telefoon altijd “Ring ring!”.

Voorbeeld:

---

```
> number (callerID billbook)
[32, 444, 123]

> receive (callerID billbook) (phone jeb)
caller ID: Jebediah
Ring ring!

> receive (callerID billbook) (phone val)
caller ID: 32 444 126
Ring ring!
```

---

## 5 Een telefoonoproep

In deze sectie wordt een programma geschreven om het begin van een telefoongesprek te simuleren: het programma vraagt naar de naam van de persoon die opgebeld moet worden, en zoekt deze naam op in het telefoonboek. Als de persoon gevonden wordt, dan wordt zijn of haar telefoonnummer gezocht en opgebeld.

### Opgave 6

Implementeer de functie `call :: PhoneBook -> [Telephone] -> IO ()`

1. Druk de tekst “Who would you like to call?” af, en lees de naam van de standaard-invoer.
2. Zoek deze `Name` op in het meegegeven `PhoneBook`. Als de naam niet in het telefoonboek staat, print dan “No such entry!” en stop het programma.
3. Zoek in de lijst met telefoons naar de `Telephone` die met het telefoonnummer van de `Entry` overeenkomt. Als een dergelijke telefoon niet bestaat, druk dan het bericht “The number you dialed does not exist.” en stop.
4. voer `receive` uit op deze `Telephone`, met als argument het nummer van de eigenaar van het *telefoonboek*.

Example:

---

```
> call billbook (map callerID [billbook,bobbook])
Who would you like to call?
Bob
Ring ring!
caller ID: Bill

> call jebbook (map callerID [billbook,bobbook])
Who would you like to call?
Valentina
The number you dailed does not exist!

> call billbook telephones
Who would you like to call?
Valentina
No such entry!

> call jebbook telephones
Who would you like to call?
Valentina
Ring ring!
caller ID: Jebediah
```

---

## 6 Alternatieve Index: Lookup

In deze sectie moet een `Index`-instance gedefiniëerd worden voor een ander datatype dan `Assoc`, namelijk `Lookup`. Het datatype heeft de volgende definitie:

---

```
data Lookup k = MkLookup (k -> Maybe Entry)
```

---

Een `Lookup` bevat enkel een functie van het type `k -> Maybe Entry`, die bepaalt welke `Entry` bij welke key hoort.

### Opgave 7

Vervolledig de `Index`-instance voor `Lookup`. Voorbeelden:

---

```
> findEntry "Bill" (empty :: Lookup String)
Nothing

> findEntry "Bob" (singleton "Bob" bob :: Lookup String)
Just (MkEntry "Bob" [32,444,124])

> let sBob = singleton (name bob) bob
> let sJeb = singleton (name jeb) jeb
> findEntry "Jebediah" (sBob <+> sJeb :: Lookup String)
Just (MkEntry "Jebediah" [32,444,125])
```

---