

INTRODUCTION TO

FUNCTIONAL PROGRAMMING

OUTLINE

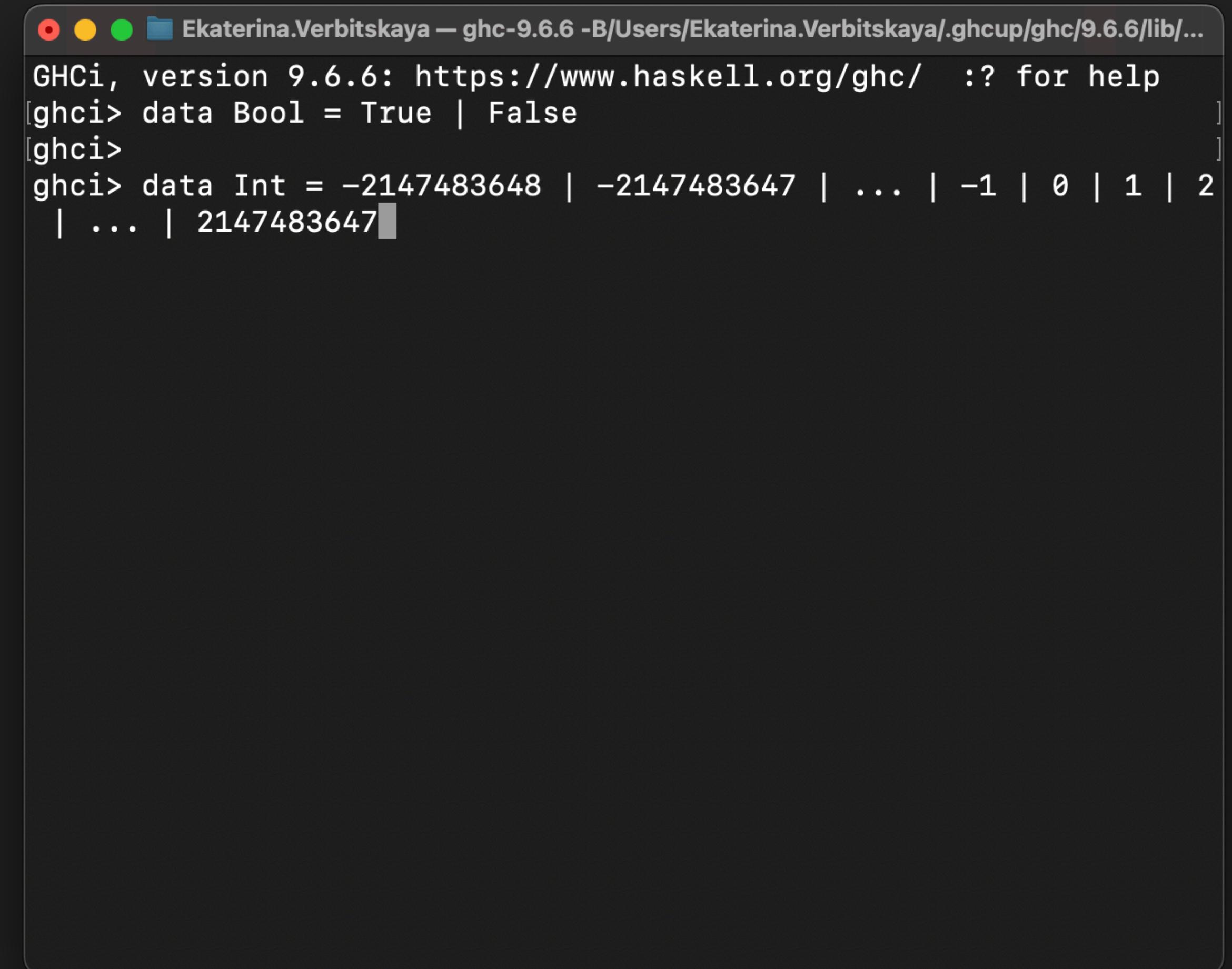
- ▶ Algebraic Data Types
- ▶ Type Classes
- ▶ Recursive Data Types

WHAT IS A TYPE?

- ▶ A collection of its values
- ▶ Bool === { True, False }
- ▶ Int === { -2147483648, -2147483647, ..., 0, 1, ..., 2147483647 }
- ▶ [Bool] === { [], [True], [False], [True, True], [True, False], [False, True], ... }
- ▶ a -> a === { \x -> x }

STANDARD TYPES AS ADT

- ▶ **Bool, Int** – type name
- ▶ **True, False, 2147483647** – values constructors
- ▶ **|** – “or”
- ▶ (**Int** is not defined like this in reality)



A screenshot of a terminal window showing a Haskell GHCi session. The title bar says "Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...". The session starts with "GHCi, version 9.6.6: https://www.haskell.org/ghc/ :? for help". Then it defines the Bool type with "data Bool = True | False". It defines the Int type with "data Int = -2147483648 | -2147483647 | ... | -1 | 0 | 1 | 2 | ... | 2147483647". The last line of the session is partially visible.

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/ :? for help
[ghci> data Bool = True | False
[ghci>
ghci> data Int = -2147483648 | -2147483647 | ... | -1 | 0 | 1 | 2
| ... | 2147483647]
```

LET'S REVISIT SHAPEAREA

- ▶ How many issues can you spot in this code?

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci| shapeArea (shape, a, b) =
[ghci|   case shape of
[ghci|     "square" -> a * b
[ghci|     "cone" -> pi * a * (a + sqrt (b^2 + a^2))
[ghci|     "cylinder" -> 2 * pi * b * (a + b)
[ghci|   :}
[ghci|>
[ghci| shapeArea ("square", 1, 1)
1.0
[ghci| shapeArea ("cone", 1, 2)
10.166407384630519
[ghci| shapeArea ("cylinder", 1, 2)
37.69911184307752
ghci| ]
```

LET'S REVISIT SHAPEAREA

- ▶ How many issues can you spot in this code?
- ▶ Why has a square two sides?
- ▶ What are **a** and **b**?
- ▶ What about case sensitivity?
- ▶ What if we get a “rectangle” or other string?

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci| shapeArea (shape, a, b) =
[ghci|   case shape of
[ghci|     "square" -> a * b
[ghci|     "cone" -> pi * a * (a + sqrt (b^2 + a^2))
[ghci|     "cylinder" -> 2 * pi * b * (a + b)
[ghci|   :]
[ghci| 
[ghci| shapeArea ("square", 1, 1)
1.0
[ghci| shapeArea ("cone", 1, 2)
10.166407384630519
[ghci| shapeArea ("cylinder", 1, 2)
37.69911184307752
ghci| ]
```

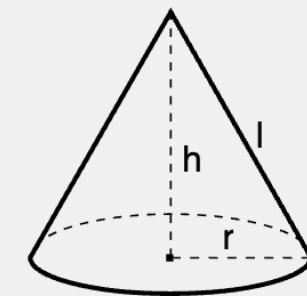
LET'S REVISIT SHAPEAREA

- ▶ How many issues can you spot in this code?
- ▶ Why has a square two sides?
- ▶ What are **a** and **b**?
- ▶ What about case sensitivity?
- ▶ What if we get a “rectangle” or other string?
- ▶ Stop, it computes wrong values

```

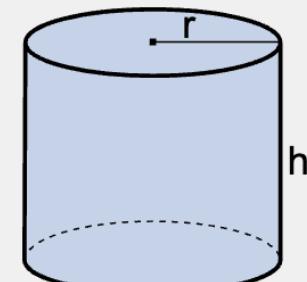
GHCi, version 9.6.6: https://www.haskell.org/ghc/9.6.6/
ghci> :{
ghci| shapeArea (shape, a, b) =
ghci|   case shape of
ghci|     "square" -> a * b
ghci|     "cone" -> pi * a * (a + sqrt(b))
ghci|     "cylinder" -> 2 * pi * b *
ghci|   :
ghci>
ghci> shapeArea ("square", 1, 1)
1.0
ghci> shapeArea ("cone", 1, 2)
10.166407384630519
ghci> shapeArea ("cylinder", 1, 2)
37.69911184307752
ghci>

```



Radius (r)	1 cm
Height (h)	2 cm
Slant height (l)	2.236 cm

Results	
Surface area (A)	10.166 cm ²
Volume (V)	2.0944 cm ³
Lateral surface area (A_L)	7.025 cm ²
Base area (A_B)	3.1416 cm ²



Base radius (r)	1 cm
Height (h)	2 cm
Surface areas	
Base	6.283185 cm ²
Lateral	12.56637 cm ²
Total	18.849556 cm ²

BETTER WAY TO DESIGN SHAPE

- ▶ A square is square at last
- ▶ No other shape can be created and passed to the function
- ▶ It's still hard to distinguish between **r** and **h**

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data Shape
[ghci|   = Square Double
[ghci|   | Cone Double Double
[ghci|   | Cylinder Double Double
[ghci|
[ghci| shapeArea :: Shape -> Double
[ghci| shapeArea (Square a) = a^2
[ghci| shapeArea (Cone r h) = pi * r * (r + sqrt (r^2 + h^2))
[ghci| shapeArea (Cylinder r h) = 2 * pi * r * (r + h)
[ghci| :}
[ghci>
[ghci> shapeArea (Square 1)
1.0
[ghci> shapeArea (Cone 1 2)
10.166407384630519
[ghci> shapeArea (Cylinder 1 2)
18.84955592153876
ghci>
```

PROPERTIES OF ADTS

▶ Distinctness

- ▶ $\forall j \neq i . C_i^n(x) \neq C_j^n(y)$

▶ Injectivity

- ▶ $C_i^n(x_1, \dots, x_n) = C_j^n(y_1, \dots, y_n) \Rightarrow \forall k . x_k = y_k$

▶ Exhaustiveness

- ▶ $x \text{ of ADT} \Rightarrow \exists i . x = C_i^n(y_1, \dots, y_n)$

▶ Selection

- ▶ $\exists s_i^k . s_i^n(C_k^n(x_1, \dots, x_n)) = x_i$

```

Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data Shape
[ghci|   = Square Double
[ghci|   | Cone Double Double
[ghci|   | Cylinder Double Double
[ghci|
[ghci| shapeArea :: Shape -> Double
[ghci| shapeArea (Square a) = a^2
[ghci| shapeArea (Cone r h) = pi * r * (r + sqrt (r^2 + h^2))
[ghci| shapeArea (Cylinder r h) = 2 * pi * r * (r + h)
[ghci| :}
[ghci>
[ghci> shapeArea (Square 1)
1.0
[ghci> shapeArea (Cone 1 2)
10.166407384630519
[ghci> shapeArea (Cylinder 1 2)
18.84955592153876
ghci> █

```

FAILING COMPUTATIONS: MAYBE

- ▶ `data Maybe a = Just a | Nothing`
 - ▶ `from Data.Maybe`
- ▶ A way to fix partiality of a function
- ▶ Only use it when there is a single way for a function to fail
- ▶ `isJust`, `isNothing`, `fromJust`, `fromMaybe`, `listToMaybe`, `catMaybes`, `mapMaybes`, `maybe`

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci]> :m Data.Maybe
[ghci]> :{
[ghci| safeHead (h:_)= Just h
[ghci| safeHead []= Nothing
[ghci|
[ghci| :}
[ghci]> safeHead [1,2,3]
Just 1
[ghci]> safeHead []
Nothing
[ghci]> :{
[ghci| collectHeads xs =
[ghci|   map fromJust $ filter isJust $ map safeHead xs
[ghci| :}
[ghci]> collectHeads [[], [1,2,3], [4], [], [5,6], []]
[1,4,5]
[ghci]> collectHeads = mapMaybe safeHead
[ghci]> collectHeads [[], [1,2,3], [4], [], [5,6], []]
[1,4,5]
[ghci]> :t mapMaybe
mapMaybe :: (a -> Maybe b) -> [a] -> [b]
ghci> █
```

FAILING COMPUTATIONS: EITHER

- ▶ `data Either a b = Left a | Right b`
- ▶ from `Data.Either`
- ▶ Right for the right answer
- ▶ Left for fail
- ▶ `lefts`, `rights`, `isLeft`, `isRight`, `fromLeft`, `fromRight`, `partitionEithers`

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci]> :m + Data.Either
[ghci]> :{
[ghci| validatePassword oldPwds minLen pwd
[ghci|   | length pwd < minLen =
[ghci|     | Left $ "Password must be longer than " ++ show minLen
[ghci|     | pwd `elem` oldPwds =
[ghci|       | Left "Password must not match previously used ones"
[ghci|       | otherwise = Right pwd
[ghci| validatePwds =
[ghci|   map (validatePassword ["pass", "word"] 4)
[ghci| chooseValidPwds =
[ghci|   rights . validatePwds
[ghci| whyPwdsNotCorrect =
[ghci|   lefts . validatePwds
[ghci| :}
[ghci]> chooseValidPwds ["pwd", "pass", "password", "wrd"]
["password"]
[ghci]> whyPwdsNotCorrect ["pwd", "pass", "password", "wrd"]
["Password must be longer than 4", "Password must not match previous
ly used ones", "Password must be longer than 4"]
ghci>
```

EXERCISES

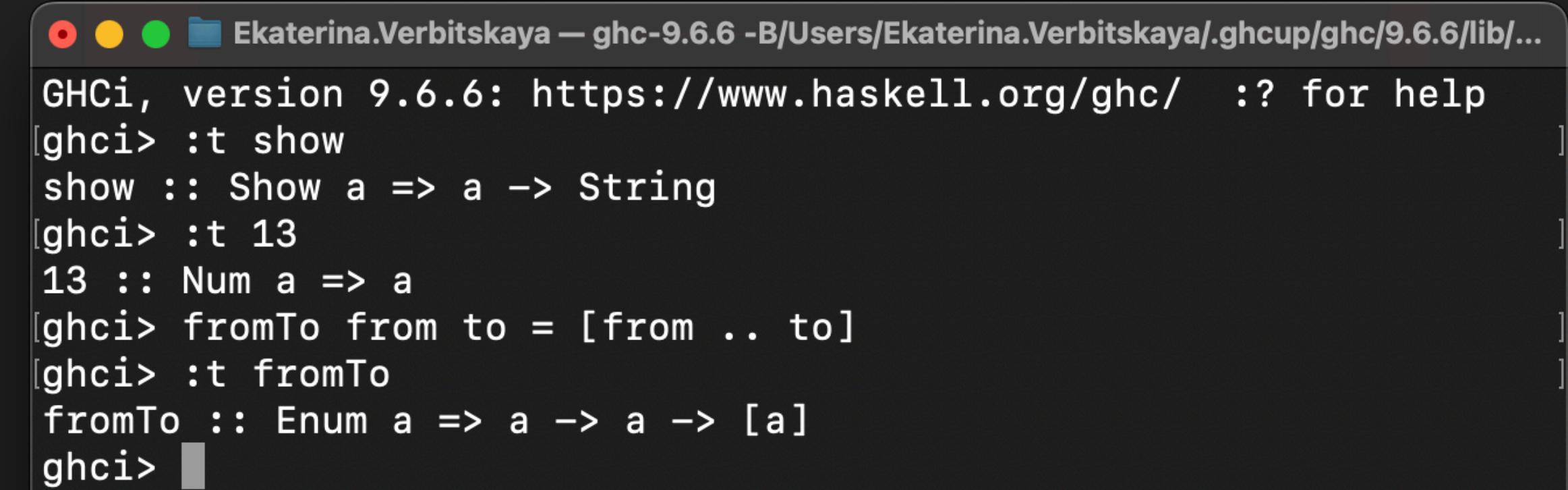
- ▶ Fix partially applied functions in HW01: pick the best suiting way to represent failing computations

OUTLINE

- ▶ Algebraic Data Types
- ▶ Type Classes
- ▶ Recursive Data Types

WE'VE SEEN THEM BEFORE

- ▶ Everything before `=>` is a constraint
- ▶ Describes behaviour through available functions



A screenshot of a terminal window titled "Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...". The window shows a GHCi session with the following interactions:

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  ?: for help
[ghci> :t show
show :: Show a => a -> String
[ghci> :t 13
13 :: Num a => a
[ghci> fromTo from to = [from .. to]
[ghci> :t fromTo
fromTo :: Enum a => a -> a -> [a]
ghci> ]
```

CLASS DEFINITION SYNTAX

- ▶ `{-# MINIMAL ... #-}` describes which functions should be implemented in any instance
- ▶ Then goes a list of functions of a type class
- ▶ Some (or all) functions can have default implementations

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| class Show a where
[ghci|   {-# MINIMAL showsPrec | show #-}
[ghci|     showPrec :: Int -> a -> ShowS
[ghci|     show    :: a    -> String
[ghci|     showList :: [a] -> ShowS
[ghci|
[ghci|     showPrec _ x s = show x ++ s
[ghci|     show x        = showPrec 0 x ""
[ghci|     showList ls s = showList_ (showPrec 0) ls s
[ghci|
[ghci|     showList_ :: (a -> ShowS) -> [a] -> ShowS
[ghci|     showList_ []      s = "[]" ++ s
[ghci|     showList_ showx (x:xs) s = '[' : showx x (showl xs)
[ghci|       where
[ghci|         showl []      = ']' : s
[ghci|         showl (y:ys) = ',' : showx y (showl ys)
[ghci| :}
```

INSTANCES

- ▶ When you type an expression into ghci, it calls **show** on the expression, so it assumes **Show**
- ▶ You need to provide an instance – the implementation of **Show**
- ▶ You need to implement at least minimal functions, but you can implement all functions of a type class

```

Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci]> :m Text.Printf
[ghci]> :{
[ghci| data Shape = Square Double
[ghci|           | Cone Double Double
[ghci|           | Cylinder Double Double
[ghci| :}
[ghci]> Square 1

<interactive>:7:1: error: [GHC-3999]
  • No instance for 'Show Shape' arising from a use of 'print'
  • In a stmt of an interactive GHCi command: print it
[ghci]> :{
[ghci| instance Show Shape where
[ghci|   show (Square a) = printf "Square %s" (show a)
[ghci|   show (Cone r h) =
[ghci|     printf "Cone with r=%s h=%s" (show r) (show h)
[ghci|   show (Cylinder r h) =
[ghci|     printf "Cylinder with r=%s h=%s" (show r) (show h)
[ghci| :}
[ghci]> Square 1
Square 1.0
[ghci]> Cylinder 13 42
Cylinder with r=13.0 h=42.0
[ghci]>

```

EQ TYPE CLASS

- ▶ Not everything can be checked for equality, only instances of `Eq`
- ▶ See [documentation](#)

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci| data Shape = Square Double
[ghci|           | Cone Double Double
[ghci|           | Cylinder Double Double
[ghci| :}
[ghci| Square 1 == Square 2
<interactive>:6:10: error: [GHC-3999]
  • No instance for 'Eq Shape' arising from a use of '=='
  • In the expression: Square 1 == Square 2
    In an equation for 'it': it = Square 1 == Square 2
[ghci| :{
[ghci| instance Eq Shape where
[ghci|   Square x == Square y = x == y
[ghci|   Cone r h == Cone r1 h1 = r == r1 && h == h1
[ghci|   Cylinder r h == Cylinder r1 h1 = r == r1 && h == h1
[ghci|   _ == _ = False
[ghci| :}
[ghci| Square 1 == Square 2
False
[ghci| Cone 1 2 == Cylinder 1 2
False
[ghci| Cone 1 2 == Cone 1 2
True
```

EQ TYPE CLASS

- ▶ Not everything can be checked for equality, only instances of **Eq**
- ▶ See [documentation](#)
- ▶ What will happen when I enter the expression **Square 1 < Square 2?**

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci|   data Shape = Square Double
[ghci|           | Cone Double Double
[ghci|           | Cylinder Double Double
[ghci| :}
[ghci| :{
[ghci|   instance Eq Shape where
[ghci|     Square x == Square y = x == y
[ghci|     Cone r h == Cone r1 h1 = r == r1 && h == h1
[ghci|     Cylinder r h == Cylinder r1 h1 = r == r1 && h == h1
[ghci|     _ == _ = False
[ghci| :}
[ghci| Square 1 == Square 1
[ghci| True
[ghci| Square 1 < Square 2]
```

EQ TYPE CLASS

- ▶ Not everything can be checked for equality, only instances of **Eq**
- ▶ See [documentation](#)
- ▶ What will happen when I enter the expression **Square 1 < Square 2?**
- ▶ Error, because **Eq** has nothing to do with comparisons

```

Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data Shape = Square Double
[ghci|           | Cone Double Double
[ghci|           | Cylinder Double Double
[ghci| :}
[ghci> :{
[ghci| instance Eq Shape where
[ghci|   Square x == Square y = x == y
[ghci|   Cone r h == Cone r1 h1 = r == r1 && h == h1
[ghci|   Cylinder r h == Cylinder r1 h1 = r == r1 && h == h1
[ghci|   _ == _ = False
[ghci| :}
[ghci> Square 1 == Square 1
True
[ghci> Square 1 < Square 2

```

<interactive>:14:10: error: [GHC-3999]

- No instance for 'Ord Shape' arising from a use of '<'
- In the expression: Square 1 < Square 2
 - In an equation for 'it': it = Square 1 < Square 2

ghci>

ORD TYPE CLASS

- ▶ **Ord**: when you want to compare stuff
- ▶ See [documentation](#)
- ▶ Let's make a custom **Pair** an instance of **Ord**

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data Pair a b = Pair a b
[ghci|
[ghci| instance (Ord a, Ord b) => Ord (Pair a b) where
[ghci|   Pair x y <= Pair x' y' = x <= x' && y <= y'
[ghci| :} ]
```

ORD TYPE CLASS

- ▶ **Ord**: when you want to compare stuff
- ▶ See [documentation](#)
- ▶ Let's make a custom **Pair** an instance of **Ord**
- ▶ Oops: we need to make it an instance of **Eq**

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data Pair a b = Pair a b
[ghci|
[ghci| instance (Ord a, Ord b) => Ord (Pair a b) where
[ghci|   Pair x y <= Pair x' y' = x <= x' && y <= y'
[ghci| :}

<interactive>:4:10: error: [GHC-3999]
  • Could not deduce 'Eq (Pair a b)'
    arising from the superclasses of an instance declaration
    from the context: (Ord a, Ord b)
    bound by the instance declaration at <interactive>:4:10-4
1
  • In the instance declaration for 'Ord (Pair a b)'
ghci>
```

ORD TYPE CLASS

- ▶ **Ord**: when you want to compare stuff
- ▶ See [documentation](#)
- ▶ Let's make a custom **Pair** an instance of **Ord**
- ▶ Oops: we need to make it an instance of **Eq**

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data Pair a b = Pair a b
[ghci|
[ghci| instance (Eq a, Eq b) => Eq (Pair a b) where
[ghci|   Pair x y == Pair x' y' = x == x' && y == y'
[ghci|
[ghci| instance (Ord a, Ord b) => Ord (Pair a b) where
[ghci|   Pair x y <= Pair x' y' = x <= x' && y <= y'
[ghci| :}
[ghci> Pair 1 2 == Pair 2 3
False
[ghci> Pair 1 2 <= Pair 2 3
True
[ghci> Pair (-1) 2 <= Pair 2 (-1)
False
[ghci> Pair 2 3 > Pair 1 2
True
ghci>
```

BOUNDED, ENUM

▶ Bounded

▶ Enum

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> maxBound :: Int
9223372036854775807
[ghci> maxBound :: Char
'\1114111'
[ghci> maxBound :: Bool
True
[ghci> succ False
True
[ghci> succ True
*** Exception: Prelude.Enum.Bool.succ: bad argument
[ghci> pred True
False
[ghci> pred False
*** Exception: Prelude.Enum.Bool.pred: bad argument
[ghci> succ 'a'
'b'
ghci> ]
```

SOME TYPES ARE NOT INSTANCES OF SOME TYPE CLASSES

- ▶ **String** (in fact, any **List**) is not an instance of either **Enum** or **Bounded**
- ▶ Strings can have any length, so they are not bounded
- ▶ What is the 'next' string?
- ▶ When it doesn't make sense, don't force the instance

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> "a" < "b"
True
[ghci> "a" < "aa"
True
[ghci> succ "a"

<interactive>:3:1: error: [GHC-3999]
  • No instance for 'Enum String' arising from a use of 'succ'
    • In the expression: succ "a"
      In an equation for 'it': it = succ "a"
[ghci>
[ghci> maxBound :: String

<interactive>:5:1: error: [GHC-3999]
  • No instance for 'Bounded String' arising from a use of 'max
Bound'
    • In the expression: maxBound :: String
      In an equation for 'it': it = maxBound :: String
ghci> █
```

DERIVING

- ▶ Some instances are boilerplate, for which an automatic deriving is possible
- ▶ `Show`, `Eq`, `Ord`, `Bounded`, `Enum` can be derived

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci|   data Weekday
[ghci|     = Monday
[ghci|     | Tuesday
[ghci|     | Wednesday
[ghci|     | Thursday
[ghci|     | Friday
[ghci|   deriving (Show, Eq, Ord, Bounded, Enum)
[ghci| :}
[ghci| Monday
Monday
[ghci| succ Monday < Thursday
True
[ghci| maxBound :: Weekday
Friday
ghci| ]
```

EXERCISES

- ▶ Make custom error data types for functions from the previous exercise, make them an instance of `Show`
- ▶ Make expression data type an instance of `Eq` and `Ord`

OUTLINE

- ▶ Algebraic Data Types
- ▶ Type Classes
- ▶ Recursive Data Types

LIST

- ▶ We can use type constructor in the definition of the type
- ▶ Nothing is that different, compared to non-recursive data structures

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci|   data List a
[ghci|     = Nil
[ghci|     | Cons a (List a)
[ghci|
[ghci|   instance Show a => Show (List a) where
[ghci|     show Nil = "[]"
[ghci|     show (Cons h t) = show h ++ " : " ++ show t
[ghci|
[ghci|   list = Cons 13 (Cons 42 Nil)
[ghci| :}
[ghci| 
[ghci| list
[ghci| 13 : 42 : []
[ghci| ]
```

LIST

- ▶ We can use type constructor in the definition of the type
- ▶ Nothing is that different, compared to non-recursive data structures
- ▶ Well, you cannot derive some instances

```
Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/...
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| data List a
[ghci|   = Nil
[ghci|   | Cons a (List a)
[ghci|   deriving (Show, Eq, Ord, Bounded, Enum)
[ghci| :}

<interactive>:5:28: error: [GHC-58291]
  • Can't make a derived instance of 'Bounded (List a)':
    'List' must be an enumeration type
    (an enumeration consists of one or more nullary, non-GADT
constructors)
    or
    'List' must have precisely one constructor
  • In the data declaration for 'List'

<interactive>:5:37: error: [GHC-30750]
  • Can't make a derived instance of 'Enum (List a)':
    'List' must be an enumeration type
    (an enumeration consists of one or more nullary, non-GADT
constructors)
  • In the data declaration for 'List'
ghci>
```

EXERCISES

- ▶ Implement 5 of any list functions from [Prelude](#) for our [List](#)
- ▶ Create any 2 instances of any type classes for our [List](#)