```
ouzo2:code$ ghci
GHCi, version 8.2.2: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /Users/hallgren/.ghci
Prelude> [0.5, 1, .. 3]

<interactive>:1:10: error: parse error on input '..'
Prelude> [0.5, 1 .. 3]
[0.5,1.0,1.5,2.0,2.5,3.0]
Prelude> [0.5, 1 .. 4]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0]
Prelude> [0.3, 1 .. 4]
[0.3,1.0,1.7,2.4,3.0999999999999996,3.7999999999999994]
Prelude> minBound ::Int
-9223372036854775808
Prelude> maxBound ::Int
9223372036854775807
Prelude> :l Overloading.hs
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> example_hand_3
Add (Card {rank = Numeric 5, suit = Hearts}) (Add (Card {rank = Ace, suit = Spad
es}) (Add (Card {rank = King, suit = Clubs}) Empty))
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> Spades
♠
*Overloading> example_hand_3
Add (Card {rank = Numeric 5, suit = ♥}) (Add (Card {rank = Ace, suit = ♠}) (Add
(Card {rank = King, suit = ♣}) Empty))
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> all
all        all_ranks
*Overloading> all_ranks
[2,3,4,5,6,7,8,9,10,J,Q,K,A]
*Overloading> 5
5
*Overloading> Numeric 5
5
*Overloading> example_hand_
example_hand_0  example_hand_1  example_hand_2  example_hand_3
*Overloading> example_hand_3
Add (Card {rank = 5, suit = ♥}) (Add (Card {rank = A, suit = ♠}) (Add (Card {ran
k = K, suit = ♣}) Empty))
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> example_hand_3
Add 5♥ (Add A♠ (Add K♣ Empty))
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
```

```
*Overloading> example_hand_3
5♥ A♠ K♣
*Overloading> (example_hand_3,ex
example_card_1  example_hand_0  example_hand_2  exp
example_card_2  example_hand_1  example_hand_3  exponent
*Overloading> (example_hand_3,example_1)

<interactive>:21:17: error: Variable not in scope: example_1
*Overloading> (example_hand_3,example_hand_1)
(5♥ A♠ K♣ ,K♣ )
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )

Overloading.hs:110:27: error:
    'value' is not a (visible) method of class 'Small'
    |
110 | instance Small ()   where value = [()]
    |                           ^^^^^

Overloading.hs:112:27: error:
    'value' is not a (visible) method of class 'Small'
    |
112 | instance Small Bool where value = enumAll
    |                           ^^^^^

Overloading.hs:113:27: error:
    'value' is not a (visible) method of class 'Small'
    |
113 | instance Small Char where value = enumAll
    |                           ^^^^^
Failed, no modules loaded.
Prelude> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> values
[()]
*Overloading> values :: [Bool]
[False,True]
*Overloading> length (values :: [Char])
1114112
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> values [(Bool,Bool)]

<interactive>:29:1: error:
    • Couldn't match expected type '[(a1, b0)] -> t'
                  with actual type '[a0]'
    • The function 'values' is applied to one argument,
      but its type '[a0]' has none
      In the expression: values [(Bool, Bool)]
      In an equation for 'it': it = values [(Bool, Bool)]
    • Relevant bindings include it :: t (bound at <interactive>:29:1)
```

```
<interactive>:29:10: error: Data constructor not in scope: Bool

<interactive>:29:15: error: Data constructor not in scope: Bool
*Overloading> values :: [(Bool,Bool)]
[(False,False),(False,True),(True,False),(True,True)]
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> :r
Ok, one module loaded.
*Overloading> smallCheck1 prop_Ace
True
*Overloading> smallCheck1 prop_Face

<interactive>:35:13: error:
    • Couldn't match type 'Card -> Bool' with 'Bool'
      Expected type: Card -> Bool
        Actual type: Card -> Card -> Bool
    • Probable cause: 'prop_Face' is applied to too few arguments
      In the first argument of 'smallCheck1', namely 'prop_Face'
      In the expression: smallCheck1 prop_Face
      In an equation for 'it': it = smallCheck1 prop_Face
*Overloading> :t prop_Face
prop_Face :: Card -> Card -> Bool
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )
Ok, one module loaded.
*Overloading> smallCheck1 prop_Face
True
*Overloading> 52*52
2704
*Overloading> :t g
g :: String -> String
*Overloading> g "42"
"52"
*Overloading> :r
[1 of 1] Compiling Overloading      ( Overloading.hs, interpreted )

Overloading.hs:172:7: error:
    • Ambiguous type variable 'a0' arising from a use of 'show'
      prevents the constraint '(Show a0)' from being solved.
      Probable fix: use a type annotation to specify what 'a0' should be.
      These potential instances exist:
        instance Show Ordering -- Defined in 'GHC.Show'
        instance Show Integer -- Defined in 'GHC.Show'
        instance Show Card -- Defined at Overloading.hs:79:10
        ...plus 27 others
        ...plus 16 instances involving out-of-scope types
        (use -fprint-potential-instances to see them all)
    • In the expression: show (read s)
      In an equation for 'f': f s = show (read s)
```

```
    |
172 | f s = show (read s)          -- defaulting does not kick in
    |       ^^^^^^^^^^^^

Overloading.hs:172:13: error:
    • Ambiguous type variable ‘a0’ arising from a use of ‘read’
      prevents the constraint ‘(Read a0)’ from being solved.
      Probable fix: use a type annotation to specify what ‘a0’ should be.
      These potential instances exist:
        instance Read Ordering -- Defined in ‘GHC.Read’
        instance Read Integer -- Defined in ‘GHC.Read’
        instance Read a => Read (Maybe a) -- Defined in ‘GHC.Read’
        ...plus 22 others
        ...plus 7 instances involving out-of-scope types
        (use -fprint-potential-instances to see them all)
    • In the first argument of ‘show’, namely ‘(read s)’
      In the expression: show (read s)
      In an equation for ‘f’: f s = show (read s)
    |
172 | f s = show (read s)          -- defaulting does not kick in
    |             ^^^^^^
Failed, no modules loaded.
Prelude> readFile "example.txt"
"June is warm\nJuly is warm\nJanuary is cold\n"
Prelude> lines <$> readFile "example.txt"
["June is warm","July is warm","January is cold"]
Prelude> fmap words "abcd efgh 123"

<interactive>:45:12: error:
    • Couldn't match type ‘Char’ with ‘[Char]’
      Expected type: [String]
        Actual type: [Char]
    • In the second argument of ‘fmap’, namely ‘"abcd efgh 123"’
      In the expression: fmap words "abcd efgh 123"
      In an equation for ‘it’: it = fmap words "abcd efgh 123"
Prelude> map words "abcd efgh 123"

<interactive>:46:11: error:
    • Couldn't match type ‘Char’ with ‘[Char]’
      Expected type: [String]
        Actual type: [Char]
    • In the second argument of ‘map’, namely ‘"abcd efgh 123"’
      In the expression: map words "abcd efgh 123"
      In an equation for ‘it’: it = map words "abcd efgh 123"
Prelude> :t foldr
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
Prelude> :t sum
sum :: (Num a, Foldable t) => t a -> a
Prelude> :i Foldable
class Foldable (t :: * -> *) where
  Data.Foldable.fold :: Monoid m => t m -> m
  foldMap :: Monoid m => (a -> m) -> t a -> m
  foldr :: (a -> b -> b) -> b -> t a -> b
  Data.Foldable.foldr' :: (a -> b -> b) -> b -> t a -> b
```

```
  foldl :: (b -> a -> b) -> b -> t a -> b
  Data.Foldable.foldl' :: (b -> a -> b) -> b -> t a -> b
  foldr1 :: (a -> a -> a) -> t a -> a
  foldl1 :: (a -> a -> a) -> t a -> a
  Data.Foldable.toList :: t a -> [a]
  null :: t a -> Bool
  length :: t a -> Int
  elem :: Eq a => a -> t a -> Bool
  maximum :: Ord a => t a -> a
  minimum :: Ord a => t a -> a
  sum :: Num a => t a -> a
  product :: Num a => t a -> a
  {-# MINIMAL foldMap | foldr #-}
        -- Defined in ‘Data.Foldable’
instance Foldable [] -- Defined in ‘Data.Foldable’
instance Foldable Maybe -- Defined in ‘Data.Foldable’
instance Foldable (Either a) -- Defined in ‘Data.Foldable’
instance Foldable ((,) a) -- Defined in ‘Data.Foldable’
Prelude> sum (42,5)
5
Prelude> sum [42,5]
47
Prelude> sum (42,5)
5
Prelude> length  (42,5)
1
Prelude> :l TestDataGenerators.hs
[1 of 2] Compiling Overloading      ( Overloading.hs, interpreted )
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample arbitrary
()
()
()
()
()
()
()
()
()
()
()
()
*TestDataGenerators> sample (arbitrary::Gen Int)
0
0
4
6
8
2
11
10
-11
18
9
```

```
*TestDataGenerators> sample (arbitrary::Gen Bool)
False
True
False
False
True
True
False
True
True
True
True
*TestDataGenerators> sample (arbitrary::Gen [Int])
[]
[0]
[-2,-2]
[4,-5,-3]
[8,2,8,-5]
[]
[-11,-9,0,-9,2,-4,5,-12]
[-6,12,-4,-8,0,9,11,3,0,-11,-10,-11,0,1]
[-8]
[-14,7,-15,-11,-5]
[]
*TestDataGenerators> :t generate
generate :: Gen a -> IO a
*TestDataGenerators> generate (arbitrary::Gen Int)
22
*TestDataGenerators> generate (arbitrary::Gen Int)
8
*TestDataGenerators> generate (arbitrary::Gen Int)
1
*TestDataGenerators> sample (choose (1,6))
3
6
5
3
1
4
4
2
3
6
6
*TestDataGenerators> sample (choose (1,6))
4
1
2
6
5
6
1
4
4
```

```
4
6
*TestDataGenerators> sample (vectorOf 5 (choose (1,6))

<interactive>:65:34: error:
    parse error (possibly incorrect indentation or mismatched brackets)
*TestDataGenerators> sample (vectorOf 5 (choose (1,6)))
[2,6,1,3,1]
[2,5,3,5,6]
[3,2,6,1,2]
[4,5,3,6,3]
[5,3,3,4,4]
[2,4,3,6,3]
[1,3,1,5,5]
[4,4,6,2,3]
[3,2,3,6,5]
[6,6,5,1,5]
[1,6,3,4,3]
*TestDataGenerators> sample (list 5 (choose (1,6)))
listOf      listOf1     listToHand
*TestDataGenerators> sample (listOf (choose ('a','z')))
""
"jg"
"fha"
"thv"
"pkfbbsr"
""
"ldwkzofwca"
"wwdlivvl"
"popemdfbjrbbulj"
"h"
"efkkgyyhvql"
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rSuit
♠
♥
♣
♥
♠
♣
♦
♦
♦
♦
♦
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rRank_v1
5
9
K
```

```
J
J
9
7
10
Q
7
Q
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rNumeric
6
9
5
7
8
9
6
3
8
5
8
*TestDataGenerators> sample rFaceCard
A
K
J
J
Q
K
A
A
K
K
Q
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rRank_v2
Q
3
9
K
A
Q
10
K
2
A
10
*TestDataGenerators> :t frequency
frequency :: [(Int, Gen a)] -> Gen a
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
```

```
Ok, two modules loaded.
*TestDataGenerators> :t rRank_v3
rRank_v3 :: Gen Rank
*TestDataGenerators> sample rRank_v3
7
2
4
6
2
10
Q
8
8
2
K
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rCard
4♥
Q♣
6♥
6♠
A♦
4♦
7♥
A♥
10♦
Q♣
7♥
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample evenInteger
0
0
0
8
0
-16
16
26
4
6
-22
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample nonNegative
0
1
3
1
3
```

```
2
5
5
14
10
19
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample pairsOfEvenIntegers
(0,0)
(4,2)
(-8,4)
(-10,8)
(6,0)
(-10,-6)
(-24,-22)
(-16,0)
(-12,-16)
(-8,-2)
(-16,-16)
*TestDataGenerators> :t doTwice
doTwice :: Monad m => m b -> m (b, b)
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> :t doTwice
doTwice :: IO a -> IO (a, a)
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample pairsOfEvenIntegers
(0,0)
(-2,4)
(-2,0)
(-8,-6)
(12,-4)
(-2,20)
(6,6)
(-8,14)
(2,-4)
(-34,-22)
(-24,8)
*TestDataGenerators> sample (doTwice (doTwice rCard))
((6♥,2♠),(10♣,5♠))
((3♠,J♣),(A♥,8♦))
((Q♠,7♥),(5♥,7♣))
((8♥,4♥),(J♦,A♦))
((9♣,4♠),(5♣,9♦))
((10♠,3♣),(2♥,3♥))
((4♣,3♦),(5♠,Q♥))
((K♥,5♣),(5♥,4♥))
((J♥,10♦),(A♣,8♥))
((A♦,3♦),(J♦,10♥))
```

```
((6♠,10♣),(A♦,2♣))
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rHand_v1

7♣
9♦

A♠ 10♣ 3♠


9♣ 7♦



*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rHand_v2
7♥

10♥
K♦

6♥
A♦

5♥ 9♥ K♥
2♦ 10♥

*TestDataGenerators> sample rHand_v2
2♦ K♦
2♠
8♦
6♦
J♠ 7♦ 3♦
Q♦ 10♥
A♣

6♥
2♠

*TestDataGenerators> sample rHand_v2

J♣ 9♥
6♥
9♥
8♣
A♠ A♣
6♠

A♠ 6♦
```

```
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rHand_v2
8♥ J♦ A♥ 10♥
3♣
K♥ A♥
7♦
5♣
J♠ 2♦
4♣ 7♣ 7♦ 8♥ 7♥ 3♠ 9♠

2♦ K♦
8♠ 7♥ K♣
K♠ 6♦
*TestDataGenerators> sample rHand_v2
5♣
A♥
Q♦ 9♣
A♣
A♦ K♥
K♠ Q♥
7♦
9♦ 7♠
Q♣
A♦ J♥ K♠
A♠
*TestDataGenerators> sample rHand_v2

7♠
4♣ 7♣
8♠
3♣
8♥
9♦ 6♦ Q♠
5♥
2♦
8♠
4♣
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rHand_v2
K♣ 4♠ 6♥ A♠ 3♣ 5♦ 4♣
8♠ J♦ J♥ 6♥
2♣ 2♠ 2♦ 5♠ K♥ 9♠
Q♥ 10♥ 10♦ J♥ 3♠ 5♦ 7♠ 4♠ A♦ 7♣ 2♦ 4♦ 9♣ 10♥ 10♥ 4♦ 7♣
5♦ K♠
4♠ Q♦ J♠

8♠ 3♣ 7♣ A♥ 3♣ Q♦ 10♣ 3♥
3♥ 10♣ 7♥ 7♥ A♥
7♦ Q♣ J♦ 2♥ Q♥ 10♦ 8♠ A♠
```

```
K♥ 9♣ 4♣ 8♦ 5♣ 7♣ A♥ 9♦ 8♣ 5♥ 9♣ K♣ 8♦ 8♣ 2♣ 7♦ 7♣ 10♦ 2♦ Q♦ J♦ 6♦
*TestDataGenerators> sample rHand_v2
2♦ 4♥
7♣ A♥ 2♥ 9♣ 3♦ 9♦ J♠ 10♥
5♦ 10♥ 6♥ 6♠ 10♠ 4♦ 8♥ 8♣ J♠ 7♣ Q♣
2♥ Q♦
K♦
J♣ 5♥
10♦ 3♥
7♠ Q♦
5♣ 9♦ 8♣ 3♣ 7♣ 7♠ 3♥ A♣ 4♦
A♦ 2♣ 8♣ J♦
3♣ 8♦ 3♠ K♣ 5♠ 2♥ 4♥ 5♣ 6♥ 6♥ 2♠ A♥ 6♦
*TestDataGenerators> sample rHand_v2
J♦ 3♦ J♠ A♥ 4♠ 8♠ Q♣ J♣ 10♥ 8♠ A♠ 9♦ A♥ 4♥ J♣
Q♦ 9♠ 4♥
10♣ 9♥ A♦ 6♠ K♣



3♦ 7♣ K♠
4♥
10♥
2♣ 9♣ Q♥ 2♠ K♠
K♦
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rHand_v2
J♠ 9♠ 3♥ K♠ J♥ 9♦ A♣ 9♥
7♠
Q♠
10♥ J♦ J♥
9♣ 6♥ A♦ 9♠ 7♣ J♦ 2♦ 3♠ 4♣ A♥ Q♠ J♦
5♣ 5♠ K♠ 7♠ 9♥ 6♥ A♠
4♠ J♦ K♠ 5♠ J♥ J♦ J♠ K♥ 6♥ Q♠ 9♦ Q♣ 2♥
4♠ 5♠ 10♥ 5♠ 9♣ 8♥ 7♠
6♥ K♥ 3♦ Q♦ 10♦ 3♣ J♦ J♠ A♣ J♠ 4♥ 10♣ 7♠ 2♠ 10♣ 4♥ 3♣ A♦ 2♦ 3♣ 7♠ J♠ 2♦ 2♦
4♣ 8♥
8♥ 4♥ 7♠
*TestDataGenerators> sample rHand_v2
9♠ 8♥ 8♥ 3♥ 10♥ 7♥ 10♥ 9♣ 4♦ 3♥ J♠ A♣ 4♥ 6♥ K♣ 7♠ J♠ 3♣ A♣ 8♣ 2♥
3♠ 5♦ 8♠ Q♥
5♠

2♠ 9♥ A♣ 4♦ A♠
K♥ K♦ 3♣ 5♠
2♦ 7♠ A♣ 7♦ 8♣ 3♥ A♦ 10♦ J♦ A♣ J♣ 5♠ 8♥ 4♣ 8♦ 9♦ 3♣ J♠ 3♠ 4♣
8♦ A♣ 6♠

A♥ K♥ 2♠ 9♥ 6♥
K♠ 10♦ K♣ 9♥
*TestDataGenerators> sample rHand_v2
7♣
```

```
8♣ A♦

6♦ 2♦ 7♥ 8♦ A♥ 7♠ 10♦ 7♠ 7♦ 5♦ A♠ 9♣

4♥ K♥ K♦ 7♠ 6♦

5♦
Q♦ 6♠ 6♥ A♣ 3♥ 2♥ 2♣ J♠ 9♦ 8♠ 6♥ 7♣ A♣ 8♦ Q♥ 9♠
2♦
K♠ 7♣ Q♥ 10♣ K♠ 3♥ 2♣
*TestDataGenerators> sample rHand_v2

10♠ 4♦ J♠ 9♣ 2♦ 6♥
3♥ 4♠ 6♥ Q♣ 3♦ 2♥ 10♦ 9♦ 6♣

4♣ 5♦ 10♠ K♣
8♠ 9♥
A♣ J♦ K♦
4♣
6♦ 9♥ 10♦ 10♦ Q♠ 8♣ 4♣ 8♠
Q♦ 7♣
7♥ Q♠ 6♣ J♦ 2♣
*TestDataGenerators> :t nub
nub :: Eq a => [a] -> [a]
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )

TestDataGenerators.hs:100:21: error:
    • No instance for (Arbitrary Card)
        arising from a use of 'arbitrary'
    • In a stmt of a 'do' block: cs <- arbitrary
      In the expression:
        do cs <- arbitrary
           return (listToHand (nub cs))
      In an equation for 'rHand_v3':
          rHand_v3
            = do cs <- arbitrary
                 return (listToHand (nub cs))
    |
100 | rHand_v3 = do cs <- arbitrary
    |                     ^^^^^^^^^
Failed, one module loaded.
*Overloading> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> sample rHand_v3


3♥ 2♣ J♠ 4♣
2♠ 3♦ 7♦ A♥ 10♠
K♦ 8♥ 2♥ 3♥ A♦ 2♦ Q♦
7♦ J♦ 9♦ 3♣ 5♣ K♣
K♦
8♥ 3♣ 3♠
```

```
8♦ 5♠ 3♥ Q♣ 5♥ 2♠ 2♣ 9♠ A♦ 10♥ 4♦
8♣ 3♥ J♣ J♦ 3♦ J♥ 3♠ 6♣ 6♦ 6♠ 4♣ K♥

*TestDataGenerators> sample rHand_v3

5♠
J♠ 8♦
Q♠ 5♦
K♥ 6♥ 2♣ A♦
A♠ 8♦ 9♠ Q♦ 7♥
5♦ 10♠ J♠
9♥ 6♠ 9♣ 8♥ 4♥ 6♥ J♥ Q♦ 10♣ 8♠ 9♠ 6♣ 3♦
A♠ 4♥ K♥ K♣ 6♦
10♠ 7♣ 9♣ 10♦ 8♠ 4♦ 5♠ 9♥
Q♥ 4♣ 8♣ 9♣ 10♣ 9♦ 4♠
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> :t val
validRank  values
*TestDataGenerators> :t validRank
validRank :: Rank -> Bool
*TestDataGenerators> quickCheck validRank
+++ OK, passed 100 tests.
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> quickCheck prop_Hand
+++ OK, passed 100 tests:
13% 1
10% 3
 9% 7
 9% 0
 8% 6
 8% 4
 8% 2
 7% 10
 6% 5
 5% 9
 4% 15
 3% 14
 3% 12
 3% 11
 1% 8
 1% 31
 1% 25
 1% 13
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )

TestDataGenerators.hs:156:35: error:
    Ambiguous occurrence '==>'
    It could refer to either 'Test.QuickCheck.==>',
                             imported from 'Test.QuickCheck' at TestDataGenerato
```

```
rs.hs:14:1-22
                            (and originally defined in 'Test.QuickCheck.Propert
y')
                          or 'Overloading.==>',
                            imported from 'Overloading' at TestDataGenerators.h
s:15:1-18
                            (and originally defined at Overloading.hs:151:3-5)
    |
156 | prop_insert_1 x xs = isOrdered xs ==> isOrdered (insert x xs)
    |                                   ^^^
Failed, one module loaded.
*Overloading> :r
[1 of 2] Compiling Overloading      ( Overloading.hs, interpreted )

Overloading.hs:143:20: error:
    • Variable not in scope: (==>) :: Bool -> Bool -> t
    • Perhaps you meant '==' (imported from Prelude)
    |
143 |                     ==> (fc `cardBeats` nc)
    |                     ^^^
Failed, no modules loaded.
Prelude> :r
[1 of 2] Compiling Overloading      ( Overloading.hs, interpreted )
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> quickCheck prop_insert_1
+++ OK, passed 100 tests.
*TestDataGenerators> verboseCheck prop_insert_1
Passed:
()
[]

Passed:
()
[()]

Passed:
()
[]

Passed:
()
[(),(),()]

Passed:
()
[()]

Passed:
()
[(),(),(),(),()]

Passed:
()
```

```
[(),(),(),(),()]

Passed:
()
[()]

Passed:
()
[]

Passed:
()
[(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),()]

Passed:
()
[(),(),(),(),()]

Passed:
()
[]

Passed:
()
[(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),()]

Passed:
()
[(),()]

Passed:
()
[(),(),()]

Passed:
()
[(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),()]
```

Passed:
()
[(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),()]

Passed:
()
[(),()]

Passed:
()
[(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),()]

Passed:

()
[(),(
),(),(),(),()]

Passed:
()
[(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),()]

Passed:
()
[(),()]

Passed:
()
[(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),()]

Passed:

()
[(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),()]

Passed:
()
[(),(
),()]

Passed:
()
[(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),()]

Passed:
()
[(),(
),(),
(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),
(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),()]

Passed:
()
[(),(
),(),
(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[()]

Passed:
()
[(),(
),()]

Passed:
()
[]

Passed:
()
[(),(
),()]

Passed:
()
[(),(
),(),
(),(),()]

Passed:
()
[]

Passed:
()

```
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()
,()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()
,(),(),()]

Passed:
()
[(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()
,(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
```

),(),
(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),
(),()]

Passed:
()
[(),(
),()]

Passed:
()
[(),(
),()]

Passed:
()
[(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

Passed:
()
[(),(
),(),
(),()
,(),()]

Passed:
()
[(),(
),(),
(),(),(),(),(),(),(),(),(),()]

Passed:

```
()
[(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(
),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),
(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),(),()]

+++ OK, passed 100 tests.
*TestDataGenerators> :t pro
product               prop_Hand             properFraction
prop_Ace              prop_all_validRank_1  property
prop_Face             prop_insert_1
*TestDataGenerators> :t prop
prop_Ace              prop_all_validRank_1  property
prop_Face             prop_insert_1
prop_Hand             properFraction
*TestDataGenerators> :t prop_
prop_Ace              prop_Hand             prop_insert_1
prop_Face             prop_all_validRank_1
*TestDataGenerators> :t prop_insert_1
prop_insert_1 :: Ord a => a -> [a] -> Property
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators> quickCheck prop_insert_1
*** Gave up! Passed only 76 tests.
*TestDataGenerators> :t or
or           orderedList
*TestDataGenerators> :t orderedList
orderedList :: (Arbitrary a, Ord a) => Gen [a]
*TestDataGenerators> sca
scale       scaleFloat  scanl       scanl1      scanr       scanr1
*TestDataGenerators> scan
scanl   scanl1  scanr   scanr1
*TestDataGenerators> scanr
scanr   scanr1
*TestDataGenerators> :t scanr
scanr :: (a -> b -> b) -> b -> [a] -> [b]
*TestDataGenerators> foldr (+) 0 [1..10]
55
*TestDataGenerators> scanr (+) 0 [1..10]
[55,54,52,49,45,40,34,27,19,10,0]
*TestDataGenerators> scanl (+) 0 [1..10]
[0,1,3,6,10,15,21,28,36,45,55]
*TestDataGenerators> :r
[2 of 2] Compiling TestDataGenerators ( TestDataGenerators.hs, interpreted )
Ok, two modules loaded.
*TestDataGenerators>
```