Learn Physics with Functional Programming

—

Scott N. Walck

—

Chapter 5: Exercises

Graham Strickland

October 27, 2024

5.4 We have the function `range` with the following definition:

```
range :: Int -> [Int]
range x = if x >= 0
          then [0..x]
          else [x..0]
```

`range` returns a list containing all the integers between the argument (inclusive) and 0 in increasing order, i.e, range$(2) = 0, 1, 2$, range$(-4) = -4, -3, \ldots, 0$, and range$(0) = 0$.

We demonstrate as follows:

```
ghci> range (-4)
[-4,-3,-2,-1,0]
ghci> range 2
[0,1,2]
ghci> range (-4)
[-4,-3,-2,-1,0]
ghci> range 0
[0]
```

5.5 We have the function `null'` with the following definition:

```
import Data.Foldable
```

```
null' :: (Foldable t) => t a -> Bool
null' xs = case toList xs of
  [] -> True
  (_ : _) -> False
```

`null'` returns `True` if an argument `t` of type `a` is empty, otherwise `False`. Since we are using the `Foldable` type, we import `Data.Foldable`.

We demonstrate as follows:

```
ghci> null' []
True
ghci> null' [1, 2, 3]
False
ghci> null' [1..]
False
```

5.6 We have the function `last'` with the following definition

```
import GHC.Stack (HasCallStack)

last' :: HasCallStack => [a] -> a
last' x = head (reverse x)
```

`last'` returns the last item in an argument with type that implements `HasCallStack`, an error if the argument is empty, or hangs indefinitely if the variable has infinite length.

We demonstrate as follows:

```
ghci> last' [1, 2, 3]
3
ghci> last' ["check", "mate"]
"mate"
ghci> last' []
*** Exception: Prelude.head: empty list
CallStack (from HasCallStack):
  error, called at libraries/base/GHC/List.hs:1646:3
    in base:GHC.List
  errorEmptyList, called at libraries/base/GHC/List.hs:85:11
    in base:GHC.List
  badHead, called at libraries/base/GHC/List.hs:81:28
    in base:GHC.List
  head, called at last.hs:4:11 in main:Main
  last', called at <interactive>:4:1 in interactive:Ghci3
```

5.7 We have the function `palindrome` with the following definition

```
import Distribution.Simple.Utils

palindrome :: String -> Bool
palindrome s = reverse (lowercase s) == lowercase s
```

`palindrome` uses the function `Distribution.Simple.Utils.lowercase` to check if the lowercase version of a string is the same as the lowercase version reversed, i.e., is the string a palindrome.

We demonstrate as follows:

```
ghci> palindrome "Radar"
True
ghci> palindrome "MadamImAdam"
True
ghci> palindrome "racecar"
True
ghci> palindrome "dog"
False
```

5.8 We find the first five elements of the infinite list $[9, 1, \ldots]$ as follows:

```
ghci> take 5 [9,1..]
[9,1,-7,-15,-23]
```

Thus we see that the first five elements are given by

$$[9, 1, \ldots] = [9, 1, -7, -15, -23, \ldots].$$

5.9 We have the function `cycle'` with the following definition

```
import GHC.Stack (HasCallStack)

cycle' :: forall a. HasCallStack => [a] -> [a]
cycle' xs = concat (repeat xs)
```

`cycle'` repeats an argument which implements `HasCallStack` an infinite number of times.

We demonstrate as follows:

```
ghci> take 10 (cycle' [4,7,8])
[4,7,8,4,7,8,4,7,8,4]
ghci> take 10 (cycle' [1])
[1,1,1,1,1,1,1,1,1,1]
```

5.10 (a) `["hello",42]` is not a valid expression, since it attempts to construct a list from elements of two different types, `String` and —Int—.

(b) `['h',"ello"]` is not a valid expression, since it attempts to construct a list from elements of two different types, `Char` and —String—.

(c) `['a','b','c']` is a valid expression.

(d) `length ['w','h','o']` is a valid expression.

(e) `length "hello"` is a valid expression.

(f) `reverse` is a valid expression, even though GHCI cannot print it.

5.10 It seems as if an arithmetic sequence will end at the last integer in the sequence before the last element in the constructor if the sequence is an integer sequence.

If it is a floating point sequence, i.e., one of the elements in the constructor was of floating point type, then the last number in the sequence will be the number in the sequence occurring after the last element in the constructor if that last element is further than the midpoint between two elements in the sequence, otherwise it will be the number occurring before.

We demonstrate as follows:

```
ghci> [0,3..7.5]
[0.0,3.0,6.0,9.0]
ghci> [0,3..7.49]
[0.0,3.0,6.0]
ghci> [0,3..7.499999999]
[0.0,3.0,6.0]
ghci> [0,3..7]
[0,3,6]
ghci> [0,3..8]
[0,3,6]
ghci> [0,3..9]
[0,3,6,9]
```