CSc 520

Principles of Programming Languages

20: Haskell — Exercises

Christian Collberg
collberg@cs.arizona.edu

Department of Computer Science
University of Arizona

Copyright © 2005 Christian Collberg
List Prefix

Write a recursive function \texttt{begin xs ys} that returns true if \texttt{xs} is a prefix of \texttt{ys}. Both lists are lists of integers. Include the type signature.

\texttt{> begin [] []}
True
\texttt{> begin [1] []}
False
\texttt{> begin [1,2] [1,2,3,4]}
True
\texttt{> begin [1,2] [1,1,2,3,4]}
False
\texttt{> begin [1,2,3,4] [1,2]}

List Containment

Write a recursive function `subsequence xs ys` that returns true if `xs` occurs anywhere within `ys`. Both lists are lists of integers. Include the type signature.

Hint: reuse `begin` from the previous exercise.

```haskell
> subsequence [] []
True
> subsequence [1] []
False
> subsequence [1] [0,1,0]
True
> subsequence [1,2,3] [0,1,0,1,2,3,5]
True
```
Consider the following function:

\[
mystery :: [a] -> [[a]]
mystery [] = [[]]
mystery (x:xs) = sets ++ (map (x:) sets)
\]

where sets = mystery xs

What would \(\text{mystery} [1,2]\) return? \(\text{mystery} [1,2,3]\)?

What does the function compute?
Explain what the following expressions involving foldr do:

1. `foldr (:) [] xs`
2. `foldr (:) xs ys`
3. `foldr ( y ys -> ys ++ [y]) [] xs`
Define a function \texttt{shorter \texttt{xs} \texttt{ys}} that returns the shorter of two lists.

\begin{verbatim}
> shorter [1,2] [1]
[1]
> shorter [1,2] [1,2,3]
[1,2]
\end{verbatim}
Write function `stripEmpty xs` that removes all empty strings from `xs`, a list of strings.

```haskell
> stripEmpty ["", "Hello", ", ", ", "World!""]
["Hello","World!"

> stripEmpty [""]
[]

> stripEmpty []
[]
```
Write function `merge` of two ordered lists `xs` and `ys` that returns an ordered list containing the elements from `xs` and `ys`, without duplicates.

```
> merge [1,2] [3,4]
[1,2,3,4]
> merge [1,2,3] [3,4]
[1,2,3,4]
> merge [1,2] [1,2,4]
[1,2,4]
```
Consider the following type:

```haskell
data Shape = Circle Float | Rectangle Float Float
```

Define a function `shapeLength` that computes the length of the perimeter of a shape.

Add an extra constructor to `Shape` for triangles.

Define a function which decides whether a shape is regular: a circle is regular, a square is a regular rectangular, and being equilateral makes a triangle regular.
Function Composition

Rewrite the expression

\[ \text{map } f \ (\text{map } g \ \text{xs}) \]

so that only a single call to \text{map} is used.
Let the Haskell function reduce be defined by

\[
\begin{align*}
    \text{reduce } f \; [] & \; v = v \\
    \text{reduce } f \; (x:x:s) & \; v = f \; x \; (\text{reduce } f \; x:s \; v)
\end{align*}
\]

Reconstruct the Haskell functions length, append, filter, and map using reduce. More precisely, complete the following schemata (in the simplest possible way):

\[
\begin{align*}
    \text{mylength } xs & \; = \; \text{reduce } ____ \; xs \; ____ \\
    \text{myappend } xs \; ys & \; = \; \text{reduce } ____ \; xs \; ____ \\
    \text{myfilter } p \; xs & \; = \; \text{reduce } ____ \; xs \; ____ \\
    \text{mymap } f \; xs & \; = \; \text{reduce } ____ \; xs \; ____
\end{align*}
\]