

COMP2209

Haskell Challenges

Testing and Development

ISAAC KLUGMAN

1 Development

All development was completed within Sublime using Haskell 7.6.3. Stack was used to manage and test the project throughout development. My development strategy followed a modular approach, every function has one purpose and I have avoided making any function verbose by limiting the number of lines per function.

When presented the opportunity, I have chose to write generalised functions; that is, the functions type will used Generalised Algebraic Datatypes (GADTs). The advantage of opting for this strategy is that code can be reused later during development in different contexts, this saves space and time.

Throughout development I have also dedicated time to ensure the code is readable as readability is an important aspect to programming in any language. I have opted to avoid the use of exaggerated parentheses `()` largely by using function composition `.` and function application `$`, as code is personally more readable and comprehensive when written with only one or two pairs of brackets.

2 Testing

As mentioned in the previous section, Stack was used to manage and test the project through out development. Stack has been a useful resource for separating the project code and testing code. I have wrote a unit test for each challenge, which feeds each challenge a set of inputs, which are then compared to a set of expected outputs.

Each challenge has an assertion function which compares the desired and actual output, and prints information about both to the screen. Each challenge then also has a challenge test function, which calls its assertion function on a series of inputs and desired outputs. I found this method to be very effective, by presenting each function with a range of inputs, if any were to fail it is easy to spot similarities in the inputs that cause a failure.

I have also used libraries such as QuickCheck (not used in `Test.hs`) for property testing for some of the smaller utility functions. However I have not included them in the file as `Challenges.hs` is required to export specific functions for submission, so the testing file would not be able to access them. QuickCheck has been very useful for checking a broad range of inputs without having to manually prepare them, which can lead to issues being found that otherwise would have been un-noticed.

References

[1] Remove Duplicates from a list

```
1 rmdups :: (Ord a) => [a] -> [a]
2 rmdups = map head . group . sort
```

<https://stackoverflow.com/a/16109302/10218833>

User: scvalex, 2013

[2] Insert Element to a List

```
1 insertAt :: a -> Int -> [a] -> [a]
2 insertAt newElement 0 as = newElement : drop 1 as
3 insertAt newElement i (a:as) = a : insertAt newElement (i -
  1) as
```

<https://stackoverflow.com/a/43291593/10218833>

User: Jakob Runge, 2017