Melbourne Haskell Workshop 2013

(HTML)

A 6-hour workshop intended to introduce and provide resources for working with Haskell.

This workshop is based around a central concept of domain modeling with Haskell. It uses a narrative of a physicist attempting to understand how the world works.

The workshop will cover the following topics:

Introduction	Motivation, Overview, and Approach
ADTs	Modelling with data in Haskell
Functions	Destructuring and evaluating data
Typeclasses	Code-Reuse and Laws
Ecosystem	Resources and Community
Monads	General Purpose DSLs

TODO: double check these items

Note:

If you are attending the workshop, please attempt to have the required items from the 'Resources' section available for your use.

TODO: Decide on some followup plans

Required Resources

Before you begin you will require the following...

A Text-Editor

We are assuming previous programming experience, however, if you would like a recommendation, have a look at Notepad++, Sublime Text, or the ever-popular Emacs and Vim. Just make sure that you are fluent enough to embark on exercises as they appear in the workshop.

The Haskell Platform

In order to run the programs written during this workshop you will need a Haskell installation. The easiest way to get up and running is to install the latest Haskell Platform. This is a "batteries included" installation of GHC and Cabal that includes many of the most useful packages available in the Hackage ecosystem.

Useful Resources

These resources are available to help you with any issues you face when learning Haskell:

#haskell on Freenode

An IRC channel dedicated to discussion of Haskell. This is often the easiest place to fire off a one-off question that is simple enough not to warrant a permanent listing on the internet.

Hackage

Hackage is the primary repository for Haskell packages. It is public, searchable, versioned, and uses Cabal package metadata for classification. Furthermore, this can be used to easily browse package contents, documentation and source-code.

For example, browse the Shake package and look at some of the Modules.

Hoogle

Hoogle is a Haskell module and function search-engine. Hoogle allows you to take advantage of the granular type-system used by Haskell to search not just for function-names, but for function type-signatures.

For example, have a look for the function with signature Text -> ByteString.

MFUG

MFUG is the Melbourne Functional Programmer's User Group. This group discusses many topics, including Haskell.

/r/haskell

For Reddit users, /r/haskell is a very useful resource with a great deal of information regarding recent developments in the Haskell ecosystem and community. This is a good place to ask some more advanced questions or start a flame-war.

Haskell News

Haskell News is a firehose-style haskell news aggregator taking information from sources as varied as academic-journals, and GitHub accounts.

HLint

HLint is a linting tool for Haskell source - It can often provide some useful hints about refactoring avenues for your code.

Introduction

Welcome to the Melbourne Haskell Workshop 2013.

This intent of this workshop is to provide a working introduction to Haskell for programmers in Melbourne who have not yet used the language in anger.

The following exercises will be presented as part of the workshop:

TODO: Update once decided

- Classification
- Lists
- Org-Chart
- Musical Program
- Image Program
- Web App
- Web Scraper
- Schema Language

Setup

... and Sanity Checking

Ensure that you have the following programs installed and functioning correctly:

- GHC(i)

At a command prompt, enter the following command:

```
ghci
```

This should launch the GHC Haskell REPL.

Type the following to ensure that you have a functioning REPL:

```
1 + 1
```

This should yield the result:

```
2
```

Create the following source file (program.hs):

```
main = print "hello world"
```

Compile the program as follows:

```
ghc --make program.hs
```

Run the program with the following command:

```
./program
```

The output should look as follows:

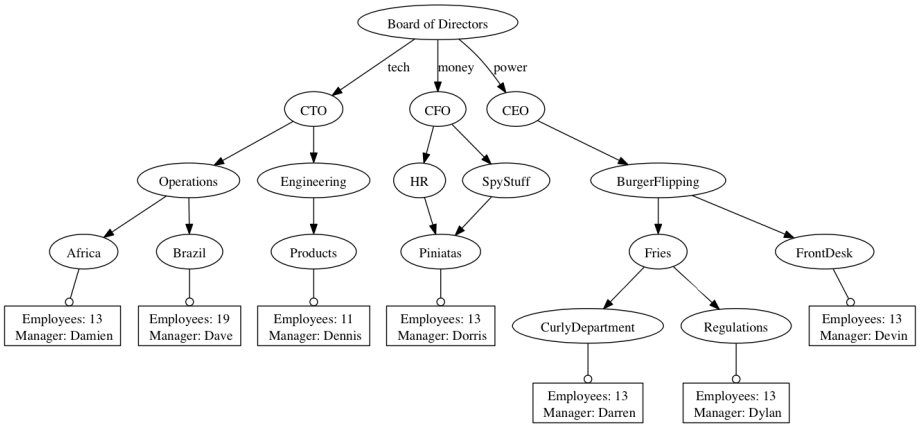
```
"hello world"
```

Exercise - Org Chart

Janice is an employee of International Banking Machines.

The CEO has asked you to model the org-chart with a computer program in order to perform some analysis on the effectiveness of middle-managers.

Before you are able to perform your calculation, you will have to decide how you will model your data. Construct a Haskell data-type to represent the various types of employees and departments shown in the following diagram.




```
digraph IBM {
    graph [ dpi = 100 ];

    b1 [label="Board of Directors"];

    b1 -> CTO [label=tech];
```

```
    b1 -> CFO [label=money];
    b1 -> CEO [label=power];
    CEO -> BurgerFlipping;
    BurgerFlipping -> Fries;
    Fries -> CurlyDepartment;
    Fries -> Regulations;
    BurgerFlipping -> FrontDesk;
    CTO -> Operations;
    CTO -> Engineering;
    CFO -> HR;
    CFO -> SpyStuff;
    Engineering -> Products;
    Operations -> Africa;
    Operations -> Brazil;
    HR -> Piniatas;
    SpyStuff -> Piniatas;

    Africa -> x1 [arrowhead=odot];
    x1 [shape=rectangle label="Employees: 13\n Manager: Damien"];

    Brazil -> x2 [arrowhead=odot];
    x2 [shape=rectangle label="Employees: 19\n Manager: Dave"];

    Products -> x3 [arrowhead=odot];
    x3 [shape=rectangle label="Employees: 11\n Manager: Dennis"];

    CurlyDepartment -> x4 [arrowhead=odot];
    x4 [shape=rectangle label="Employees: 13\n Manager: Darren"];

    Regulations -> x5 [arrowhead=odot];
    x5 [shape=rectangle label="Employees: 13\n Manager: Dylan"];

    FrontDesk -> x6 [arrowhead=odot];
    x6 [shape=rectangle label="Employees: 13\n Manager: Devin"];

    Piniatas -> x7 [arrowhead=odot];
    x7 [shape=rectangle label="Employees: 13\n Manager: Dorris"];
}
```

Exercise - Music

Once you have a model that reaches a sufficient level of complexity you will find that your data-types become recursive.

Here is a small musical target-language:

```
data Music
  = A_ -- Underscore Represents a Flat
  | A
  | A' -- Prime represents a Sharp
       -- ... B, C, D, E, F, G ...
  | Sequence [Music]
  | Sharp Music
  | Longer Integer Music
  | Higher Integer Music
```

Certainly not comprehensive enough to write Beethoven's 5th Symphony, but good enough for Baa Baa Black-Sheep.

```
intro   = Sequence [ C, C, G, G ]
run     = Sequence [ A, B, C, A ]
middle  = Sequence [ F, F, E, E ]
run2    = Sequence [ D, C, D, E ]
baa_baa = Sequence [ Longer 2  intro
                   , Higher 12 run
                   , Longer 4  G
                   , Longer 2  middle
                   , run2
                   , Longer 4  C ]
```

In order to listen to this music, create a file with the following code:

```
import Midi
import qualified Data.ByteString as BS

main = BS.writeFile "baa_baa.mid" (make_music 4 4 (Longer 4 baa_baa))
```

Your task is to write a melody using this language. Try to write and compose functions, rather than declaring one big data-structure.

Appendix

Hutton's Razor