Lab 12 - Generating Haddock Documentation

CS 1XA3

April 3rd, 2018

Generating Documentation With Haddock

- Haddock is a tool for generating professional documentation from comments in Haskell files
- It should already be installed with the Haskell Platform and accessible from the command haddock
- Everything you need to know about Haddock is documented at http://haskell-haddock.readthedocs.io/en/latest/

Generating Documentation With Haddock

- The best way to generate Haddock documentation is from a main file, consider the project template I gave you yesturday with the following files: ExprType.hs,ExprParse.hs,ExprDiff.hs,ExprTest.hs
- In this case, ExprTest.hs can serve as our main file since it imports all the other files
- From the project directory (i.e the directory containing all the source files), create a subdirectory called docs and generate HTML Docs to it with

```
haddock --html --odir=docs ExprTest.hs
```

Documenting Declerations

- The $--|$ written before a decleration is used to specify documentation for that decleration

- Example

```
-- | Parses a string into an Expr Double type
parseExprD :: String -> Expr Double
parseExprD ss = ...
```

- Also works with multi-line comments

```
{- | Parses a string into an Expr Double type
     using the Parsec package
 -}
parseExprD :: String -> Expr Double
parseExprD ss = ...
```

Documenting Decleratiоns

- You can also specify documentation after the decleration with a slightly different syntax
- Example

```
parseExprD :: String -> Expr Double
-- ^ Parses a string into an Expr Double type
parseExprD ss = case parse exprD "" ss of
      Left err -> error $ show err
      Right expr -> expr
```

Documenting Declerations

A function decleration and it's arguments can be documented explicitely like so

```
-- | Parses a string into an Expr Double type
parseExprD :: String        -- ^ input to parse
           -> Expr Double -- ^ resulting expression
```

Note: doesn't allows work with class declerations

Module Descriptions

Module desciprtions come before the module decleration and should contain specific items

```
{-|
Module      : ExprDiff
Description : Contains a type class and instances for
              differentiable expressions
Copyright   : (c) Curtis D'Alves @2018
License     : WTFPL
Maintainer  : dalvescb@mcmaster.ca
Stability   : experimental
Portability : POSIX

TODO write a longer description of the module,
containing some commentary with @some markup@.
-}
module ExprDiff where
```

Licensing

- When creating an open source project, it's important to choose an appropriate license so people are aware of the terms of use and that you don't provide a warrenty
- Suggestion: use a WTFPL license, allows people to use code how they want and protects from liability
- https://en.wikipedia.org/wiki/WTFPL

Documenting Class Methods

Class Methods are documented with the same syntax as any decleration, largely the way you would expect

```
class DiffExpr a where
  -- | Evaluate an expression given var values
  eval :: Map.Map String a -> Expr a -> a
  -- | Simplify an expression and sub in values
  simplify :: Map.Map String a -> Expr a -> Expr a
  -- | Perform partial differention w.r.t identifier
  partDiff :: String -> Expr a -> Expr a
```

Documenting DataTypes

Datatypes are documented by constructor using the after decleration syntax

```haskell
-- | A datatype for common numeric expression
data Expr a =
   Add (Expr a) (Expr a)   -- ^ Binary Addition
 | Mult (Expr a) (Expr a) -- ^ Bianry Multiplication
 | Const a                 -- ^ Value Wrapper
 | Var String              -- ^ Variable Identifier
```

Section Headers

- A module can be split up into sections and subsections by using asterisks, i.e

```
-- * Section Title
-- ** SubSection Title
-- *** SubSubSection Title
...
```

- Example: we might want to split up the ExprType module into two sections

```
-- * DataType Decleration
data Expr a = ....

-- * Miscellaneous Functions
getVars :: Expr a -> [String]
...
```

MarkUp

Haddock Markup is filled with cool features, see the documentation for the full list of annotations

- Hyperlink Identifiers: you can reference identifiers like datatypes, classes, functions or constructors by putting them in single quotes, i.e

```
-- | This module uses the 'Expr' datatype
```

- Hyperlink Modules: reference a module with double quotes

```
-- | This depends on the "ExprType" module
```

- Enumerated Lists

```
-- | This is a bulleted list:
--       * first item
--       * second item
```