Haskell 入門

Pasberth

January 4, 2014

概 要

この文書は Haskell の簡単な教科書である．これを読めば，きっと，Haskell を理解する手助けとなるであろう．

目 次

1 はじめに

1.1 ソースコード

この文書はオープンソースである．完全なソースコードは，2014 年 1 月 4 日現在，`https://github.com/pasberth/pasberth.github.io/tree/default/`

papers/brief-intro-haskell-language で公開されている．この文書自体のソースコードや，例に使用されるソースコードがそこから入手できる．

1.2 ソースコードの入手方法

もっとも簡単な方法は git を使用することである．次のようなシェルコマンドで簡単に入手できる．

```
git clone https://github.com/pasberth/pasberth.github.io.git
cd pasberth.github.io
git checkout default
cd papers/brief-intro-haskell-language
```

1.3 Haskell2010 について

Haskell2010 のオンラインレポートは http://www.haskell.org/onlinereport/haskell2010/ にある．もしなにかわからないことがあったら，これを読もう．

2 字句規約

字句規約を簡単に説明する．もしわからないことがあったら http://www.haskell.org/onlinereport/haskell2010/haskellch2.html を読もう．

2.1 リテラル

リテラルは整数，浮動小数点数，文字，文字列である．リテラルを形式的に定義することもできるが，ここでは例を見て理解してほしい．整数は，たとえば 42 である．浮動小数点数は，たとえば 3.14 である．文字は，たとえば'x' である．文字列は，たとえば"answer"である．

2.2 コメント

単行コメントは “--”から始めて行末まで続ける．これは C 言語の “//”に相当すると思えばよろしい．複行コメントは “{-”から始めて “-}”まで続ける．これは C 言語の “/*”と “*/”に相当すると思えばよろしい．

```
-- a comment
{- comments -}
```

2.3 識別子

識別子に使用できる文字は，小文字の英字から始めて，アンダースコア (_) か大文字の英字か数字を続けたものである．たとえば valid_identifier は有効な識別子だ．しかし Invalid や invalid-identifier は不正である．ident1 のように数値を続ける事もできるが，先頭に数値をおくことはできない．また，多くの言語と同じように予約語を使用する事もできないことに注意せよ．たとえば case や class は，予約語なので識別子としては不正である[1]．

2.4 データコンストラクタ

データコンストラクタに使用できる文字は，大文字の英字から始めて，アンダースコア (_) か大文字の英字か数字を続けたものである．たとえば Valid_constructor は有効な識別子だ．しかし invalid や Invalid-constructor は不正である．Constr1 のように数値を続ける事もできるが，先頭に数値をおくことはできない．

```
-- データコンストラクタの例
Just
Nothing
```

2.5 オペレータ

オペレータに使用できる文字は，“!”か “#”か “$”か “%”か “&”か “+”か “.” か “/”か “<”か “=”か “>”か “?”か “@”か “\”か “^”か “|”か “-”か “~”か “:” かである．オペレータは，これらの文字を連ねたものである．ただし，予約語は使えない．たとえば，“->”や “<-”は予約語なのでオペレータではない[2]．

```
-- オペレータの例
.
&&&
|||
```

[1] 予約語のリストは http://www.haskell.org/onlinereport/haskell2010/haskellch2.html の *2.4 Identifiers and Operators* の *reservedid* を参照せよ．

[2] 予約語のリストは http://www.haskell.org/onlinereport/haskell2010/haskellch2.html の *2.4 Identifiers and Operators* の *reservedop* を参照せよ．

2.6 データコンストラクタオペレータ

コロン (:) から始まるオペレータはデータコンストラクタオペレータである．たとえば，“:+”はデータコンストラクタオペレータである．ただし，予約語は使えない．たとえば，“::”は予約語なのでオペレータではない[3]．

```
-- データコンストラクタオペレータの例
:+
```

2.7 名前修飾

修飾された名前は，名前空間と識別子やオペレータ，データコンストラクタ，データコンストラクタオペレータなどの名前を “.”で繋いだものである．空白を含む事ができないことに注意せよ．後述するが，空白を含めると “.”が中置関数だということになってしまう．たとえば，`Prelude.flip` は修飾された名前だ．`Prelude..` は，“.”というオペレータを修飾している．`Prelude.Just` や `Prelude.:+`は有効だ．`Prelude . flip` は誤りだ．

2.8 関数適用

式を空白で繋いだものが関数適用である．ただし，式とは識別子やデータコンストラクタやリテラルや，それらをカッコで囲った物である．これらを形式的に定義する事もできるが，ここでは例を見て理解してほしい．

```
-- 関数適用の例
f x y z
f (g x)
Just "answer"
```

このとき，オペレータはそのままでは渡せないことに注意せよ．オペレータを引数として渡したり，関数の名前として使用したいときは，カッコで囲わなければならない．

```
-- 誤り
foldl1 + [1,2,3]
+ 1 2
-- 正しい
foldl1 (+) [1,2,3]
(+) 1 2
```

[3]予約語のリストは `http://www.haskell.org/onlinereport/haskell2010/haskellch2.html` の *2.4 Identifiers and Operators* の *reservedop* を参照せよ．

2.9 中置記法

オペレータは中置することができる．たとえば，次の 2 行は等しい．

```
1 + 2
(+) 1 2
```

また，識別子を中置したい場合，“‘”で囲う．たとえば，次の 2 行は等しい．

```
5 `mod` 2
mod 5 2
```

関数適用は常に中置記法より優先順位が高いことに注意せよ．いかなる場合も，関数適用より中置記法の方が優先される事はない．優先順位がわからなくなったら，この規則を思い出そう．たとえば，次の 2 行は等しい．

```
f x + g x
(f x) + (g x)
```

オペレータには優先順位が設定されている事がある．これはオペレータごとに異なる．たとえば，次の 2 行は等しい．

```
x + y * z
x + (y * z)
```

2.10 ラムダ式

ラムダ式は “\”から始め，パラメータを連ねたあと，“->”で区切って，本体を書く．本体には関数適用や中置記法が許される．これも例を見て理解してほしい．

```
\x y -> x
\x y z -> x z (y z)
```

ラムダ式を関数の引数にすることはできない．ラムダ式を引数として渡したり，関数の名前として使用したいときは，カッコで囲わなければならない．

```
-- 誤り
f \x -> x
\x -> x y -- (\x -> x) y としたい
-- 正しい
f (\x -> x)
(\x -> x) y
```

中置記法の右辺にラムダ式をおくことはできるが，中置記法の左辺にラムダ式をおくことはできない．左辺にラムダ式をおきたい場合は，カッコで囲わなければならない．

```
-- 誤り
\x -> x . f -- (\x -> x) . f としたい
-- 正しい
(\x -> x) . f
-- これはOK
f . \x -> x
```

2.11 do 記法

do 記法を説明するには，Monad の>>や>>=を定義する必要があるが，その意味論は後回しにして，とりあえず，そういう記号があるということを覚えてほしい．ここで，do 記法は，常に次の規則が成り立つ．

```
do { m ; k } == m >> k
do { x <- m ; k } == m >>= \x -> k
```

ゆえに，do 記法は，常に>>や>>=の省略形だと思えばよろしい．たとえば，

```
do { putStrLn "Please enter your first name (followed by 'enter'):"
   ; first_name <- getLine
   ; putStrLn ("Hello, " ++ first_name)
   }
```

は

```
putStrLn "Please enter your first name (followed by 'enter'):" >>
  getLine >>= \first_name ->
   putStrLn ("Hello, " ++ first_name)
```

の省略形である．ただし，結合をカッコで強調すると，

```
(putStrLn "Please enter your first name (followed by 'enter'):") >>
  (getLine >>= (\first_name ->
   putStrLn ("Hello, " ++ first_name)))
```

となる．

ただし，do{..} は，“{”と “}”で囲う代わりに，インデントであらわしてもよい．

```
do m
   k
```

do 記法を関数の引数にすることはできない．do 記法を引数として渡したり，関数の名前として使用したいときは，カッコで囲わなければならない．

```
-- 誤り
f do m
do m y -- (do m) y としたい
-- 正しい
f (do m)
(do m) y
```

中置記法の右辺に do 記法をおくことはできるが，中置記法の左辺に do 記法をおくことはできない．左辺に do 記法をおきたい場合は，カッコで囲わなければならない．

```
-- 誤り
do m $ f -- (do m) $ f としたい
-- 正しい
(do m) $ f
-- これはOK
f $ do m
```

3　式

参考文献

[1] Haskell 2010 Language Report