関数型プログラミング

第4回　モジュール

萩野　達也
hagino@sfc.keio.ac.jp

Slide URL

https://vu5.sfc.keio.ac.jp/slide/

モジュール

- Haskellプログラムはモジュール単位に分割されている

- モジュールでは以下のものが定義されている:
  - 関数
  - 変数
  - データ型

- モジュールの読み込み
  - モジュールは，利用する前に読み込む（インポート）する必要がある

```
import Module
```

echoコマンド

echo.hs

```
import System.Environment

main = do args <- getArgs
          putStr $ unwords args
```

- コマンドの与えられた引数をそのまま標準出力に出す.

```
% ghc echo.hs
...
% ./echo a b c
a b c
% ./echo This    is    a   pen.
This is a pen.
% ./echo "This    is    a    pen."
This    is    a    pen.
%
```

Main と Prelude モジュール

- Main モジュール
  - main 変数は Main モジュールに属する
  - モジュールを指定しない場合，すべてのものは Main モジュールに属する
- Prelude モジュール
  - 基本的な型，変数，関数を定義している.
  - 暗黙でシステムに読み込まれる
  - getContents, lines, unlines, ... は Prelude モジュールで定義されている

unwords 関数

```
unwords:: [String] -> String
```

- 文字列を空白は挟んでつなげる
  - `unwords ["a", "b", "c"]` → `"a b c"`
  - `unwords ["a(1,", "2,", "3)"]` → `"a(1, 2, 3)"`
  - `unwords ["This", "is", "a", "pen."]` → `"This is a pen."`
  - `unwords ["a\n", "b"]` → `"a\n b"`
  - `unwords []` → `""`

System.Environment アクション

```
getArgs:: IO [String]
```

- コマンド引数を読み込むアクション
- アクションが成功すると，文字列のリストが返される

```
getProgName:: IO String
```

- プログラム名を読み込むアクション
- アクションが成功すると，文字列が返される

練習問題4－1

- 引数に数字が与えられたときに，その合計を出力するプログラムsum.hsを完成させなさい．

sum.hs

```
import System.Environment

main = do args <- getArgs
          print $ .... args
```

- 文字列を数字にするには**read**を用います．
  - `(read "123")::Int`
  - **read**は返す値が多層型なので欲しい型を指定する必要がある場合があります．
- **リストの数字を合計するには**`sum`**を用います**．
  - `sum::[Int]->Int`
  - `sum [1,2,3,4,5]` ⇒ `19`

```
% ghc sum.hs
...
% ./sum 1 5 3
9
% ./sum 123 248
371
%
```

fgrep コマンド

- 特定の文字列を含む行だけを入力から抜き出し出力する.
  - airline-code.txtの中からJapanを含む行を抜き出す.

```
% ./fgrep Japan < airline-code.txt
NV      Air Central     AIR CENTRAL     Japan
NQ      Air Japan       AIR JAPAN       Japan
EL      Air Nippon      ANK AIR Japan
EH      Air Nippon Network Co. Ltd.     ALFA WING       Japan
DJ      AirAsia Japan   WING ASIA       Japan
HD      AIRDO   AIR DO  Japan
NH      All Nippon Airways      ALL NIPPON      Japan
...
7G      Star Flyer      STARFLYER       Japan
JW      Vanilla Air     VANILLA Japan
```

fgrep コマンドのプログラム

fgrep.hs

```
import System.Environment
import Data.List

main = do args <- getArgs
          cs <- getContents
          putStr $ fgrep (head args) cs

fgrep :: String -> String -> String
fgrep pattern cs = unlines $ filter match $ lines cs
  where
    match :: String -> Bool
    match line = any prefixp $ tails line

    prefixp :: String -> Bool
    prefixp line = pattern `isPrefixOf` line
```

- 上記のプログラムを入力しなさい.
  - airline-code.txtからJapanを含む行を抜き出しなさい
  - airを含む行はどうですか

main アクションと fgrep 関数

```
main = do args <- getArgs
          cs <- getContents
          putStr $ fgrep (head args) cs

fgrep :: String -> String -> String
fgrep pattern cs = unlines $ filter match $ lines cs
  where
    ...
```

- **main**アクション
  - **getArgs**でコマンド引数を読み**args**に束縛
  - 標準入力を読み**cs**に束縛
  - **head**で**args**の先頭を取り出し**cs**とともに**fgrep**に渡す
  - その結果を出力する
- **fgrep**関数
  - **lines**で行うごとに分け，**unlines**で行に戻す
  - **filter**は条件を満たすリストを取り出す高階関数

headとtailとfilter関数

- **head関数**
  - `head :: [a] -> a`
  - リストの先頭の要素を返す
  - `head [1, 2, 3]` → `1`
  - `head [2, 3]` → `2`
  - `head [3]` → `3`
  - `head []` → 実行時エラー

- **tail関数**
  - `tail :: [a] -> [a]`
  - リストから先頭要素を取り除いたリストを返す
  - `tail [1, 2, 3]` → `[2, 3]`
  - `tail [2, 3]` → `[3]`
  - `tail [3]` → `[]`
  - `tail []` → 実行時エラー

- filter関数
  - `filter :: (a -> Bool) -> [a] -> [a]`
  - `filter f xs`はリスト`xs`の要素`x`のうち`f x`が`True`である要素だけを集めたリストを返す
  - `filter odd [1, 2, 3, 4, 5]` → `[1, 3, 5]`
  - `filter odd [2, 4, 6, 8, 10]` → `[]`
  - `filter odd []` → `[]`
  - ただし`odd n`は`n`が奇数の時に`True`

where 節

```
fgrep :: String -> String -> String
fgrep pattern cs = unlines $ filter match $ lines cs
  where
    match :: String -> Bool
    match line = any prefixp $ tails line

    prefixp :: String -> Bool
    prefixp line = pattern `isPrefixOf` line
```

- **where**節は式の中で必要な関数などを後で定義する構文

```
式 where 定義1
         定義2
         定義3
```

  - **式**の中で**定義1**～**定義3**で定義した関数などを利用することができる.

- **fgrep**関数の中で使う**match**関数と**prefixp**関数を定義している
  - **match**や**prefixp**は**fgrep**外の**main**などでは利用できない
  - **match**や**prefixp**では**fgrep**の引数なども参照可能

match関数の実装

```
match :: String -> Bool
match line = any prefixp $ tails line

prefixp :: String -> Bool
prefixp line = pattern `isPrefixOf` line
```

- **match line**
  - **line**(たとえば**"abcd"**)の中に**pattern**(たとえば**"bc"**)が含まれているかを調べる
    - **match "abcd"**
  - **tails**によって**line**を一文字ずつ短くした文字列を作る
    - **tails "abcd" → ["abcd", "bcd", "cd", "d", ""]**
  - **any**によって**prefixp**を満たすものがあるのかを調べる
  - **prefixp**は**pattern**で始まっているかを調べる
    - **prefixp "abcd" → "bc" `isPrefixOf` "abcd" → False**
    - **prefixp "bcd" → "bc" `isPrefixOf` "bcd" → True**
    - **prefixp "cd" → "bc" `isPrefixOf` "cd" → False**

any関数，tails関数，isPrefixOf関数

- **any関数**
  - `any :: (a -> Bool) -> [a] -> Bool`
  - `any f xs`は`xs`の各要素に`f`を適用し，そのいずれかの値が`True`なら`True`を返す．すべてが`False`ならば`False`を返す．
  - `any odd [1, 2, 3, 4, 5] → True`
  - `any odd [1, 3, 5] → True`
  - `any odd [2, 4, 6] → False`
  - `any odd [3] → True`
  - `any odd [] → False`

- **Data.List.tails関数**
  - `tails :: [a] -> [[a]]`
  - `tails xs`はリスト`xs`自身，`xs`から2番目の要素以降のリスト，3番目の要素以降のリスト，‥‥をリストにして返す
  - `tails [1, 2, 3] → [[1, 2, 3], [2, 3], [3], []]`
  - `tails [1, 2] → [[1, 2], [2], []]`

- **Data.List.isPrefixOf関数**
  - `isPrefixOf :: (Eq a) => [a] -> [a] -> Bool`
  - `isPrefixOf xs ys`はリスト`xs`がリスト`ys`の先頭に一致するときに`True`
  - `xs `isPrefixOf` ys`は`isPrefixOf xs ys`と同じで2項演算子として使う方法

練習問題4－2

fgrepi.hs

```
import System.Environment
import Data.List

main = ...

fgrepi :: String -> String -> String
fgrepi pattern cs = unlines $ filter match $ lines cs
  where
    ...
```

- fgrep.hsでは大文字と小文字を区別していましたが，区別しない形でマッチする行を探して出力するfgrepi.hsを作りなさい．
  - たとえばairline-code.txtの中でjapanを含む行を抜き出せますか．
  - 大文字を小文字に変換するためには練習問題3－2の**lower**を用いなさい．

```
lower :: Char -> Char
lower 'A' = 'a'
lower 'B' = 'b'
...
lower 'Z' = 'z'
lower c = c
```

```
% ./fgrepi japan < airline-code.txt
NV      Air Central     AIR CENTRAL     Japan
NQ      Air Japan       AIR JAPAN       Japan
EL      Air Nippon      ANK AIR Japan

...
%
```

練習問題4－3

fgrepm.hs

```
import System.Environment
import Data.List

main = do args <- getArgs
          cs <- getContents
          putStr $ fgrepm args cs

fgrepm :: [String] -> String -> String
fgrepm ps cs = unlines $ filter matchAll $ lines cs
  where
    matchAll :: Sring -> Bool
    matchAll line = all match ps
      where
        ...
```

- fgrep.hsでは与えられた引数1つにマッチする行を返しましたが，複数の引数すべてを含む行だけを出力するfgrepm.hsを作りなさい．

```
% ./fgrepm Japan All < airline-code.txt
NH       All Nippon Airways       ALL NIPPON       Japan
%
```

all関数

- **all関数**
  - `all :: (a -> Bool) -> [a] -> Bool`
  - `all f xs` は `xs` の各要素にfを適用し，そのすべての値が`True`なら`True`を返す．いずれかが`False`ならば`False`を返す．
  - `all odd [1, 2, 3, 4, 5]` → `False`
  - `all odd [1, 3, 5]` → `True`
  - `all odd [2, 4, 6]` → `False`
  - `all odd [3]` → `True`
  - `all odd []` → **`True`**

- **any関数と似ている**
  - `any :: (a -> Bool) -> [a] -> Bool`
  - `any f xs` は `xs` の各要素にfを適用し，そのいずれかの値が`True`なら`True`を返す．すべてが`False`ならば`False`を返す．
  - `any odd [1, 2, 3, 4, 5]` → `True`
  - `any odd [1, 3, 5]` → `True`
  - `any odd [2, 4, 6]` → `False`
  - `any odd [3]` → `True`
  - `any odd []` → **`False`**

関数およびアクションのまとめ

		意味
`unwords`	`unwords xs`	
`sum`	`sum xs`	
`read`	`(read s)::Int`	
`any`	`any f xs`	す
`all`	`all f xs`	す
`filter`	`filter f xs`	す
`head`	`head xs`	
`tail`	`tail xs`	
`Data.List.tails`	`tails xs`	
`Data.List.isPrefixOf`	`xs `isPrefixOf` ys`	

アクション名	意味
`System.Environment.getArgs`	
`System.Environment.getProgName`	