関数型プログラミング
第8回　関数(1)

萩野　達也
hagino@sfc.keio.ac.jp

Slide URL

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

関数定義

```
関数名 パターン_A1 パターン_A2 ···· | ガード_A1 = 定義_A1
                                    | ガード_A2 = 定義_A2
                                    :
                                    :
関数名 パターン_B1 パターン_B2 ···· | ガード_B1 = 定義_B1
                                    | ガード_B2 = 定義_B2
                                    :
                                    :
```

- パターンマッチを使って関数を定義
- 関数名および変数名は識別子
  - アルファベットの小文字で始まる
  - アルファベット大文字・小文字，数字，アンダースコア，シングルクォートからなる
- 予約語は使えない
  - **case**, **class**, **data**, **default**, **deriving**, **do**, **else**, **if**, **import**, **in**, **infix**, **infixl**, **infixr**, **instance**, **let**, **module**, **newtype**, **of**, **then**, **type**, **where**, **_**

再帰呼び出し

- 関数型プログラミング言語ではfor文やwhile文のような繰り返しがないので，再帰呼び出しを使って繰り返しを実現しています．

```
factorial n = if n == 0 then 1
                        else n * factorial(n - 1)
```

factorial の再帰呼び出し

- 再帰呼び出し
  - 関数の定義において自分自身を直接・間接的に呼び出すこと
  - 分割統治の考え方に合っている
    - 難しい問題を，簡単な問題に分割し解決する
    - 分割した問題も，元の問題と同じでパラメータのみが異なる

漸化式との類似

- 漸化式で定義された数列
  - a_n を a_{n-1} や a_{n-2} などで定義する.

$$a_n = a_{n-1} + a_{n-2}$$

- そのまま再帰呼び出しで定義できる

```
fib n | n == 0     = 1
      | n == 1     = 1
      | otherwise = fib(n - 1) + fib(n - 2)
```

例題

- 1からnまでの数字を合計する関数**sumn**を定義しなさい.

- リストの合計をする**sum**を用いると

```
sumn n = sum [1..n]
```

- 1からnまでの合計は，1から$n-1$までの合計にnを加えたものなので，再帰呼び出しで書くことができる.

```
sumn n = if n == 0 then 0
                   else n + sumn(n-1)
```

練習問題8－1

comb.hs

```
import System.Environment

main = do args <- getArgs
          print $ comb (read $ args !! 0) (read $ args !! 1)

comb n r | r == 0     = 1
         | n == r     = 1
         | otherwise = ...
```

- n個からr個を選ぶ組み合わせの数${}_nC_r$を求めなさい.
  - ${}_nC_0 = {}_nC_n = 1$
  - ${}_nC_r = {}_{n-1}C_r + {}_{n-1}C_{r-1}$

```
% ghc comb.hs
...
% ./comb 10 5
252
%
```

おつりの計算

- 商店などで170円の商品を買ったとき，200円渡すと30円のおつりを返すことにですが，おつりはいろいろな形で返すことができます．
  - 10円玉3枚
  - 5円玉6枚
  - 1円玉30枚
  - 1円玉10枚と5円玉2枚と10円玉1枚

- おつりの返し方の場合の数を計算しましょう
  - $a(n)$ を n 円を1円玉で返す場合の数だとすると
    - $a(n) = 1$
  - $b(n)$ を n 円を1円玉と5円玉で返す場合の数だとすると
    - $b(n) = a(n)$ $(n < 5)$
    - $b(n) = a(n) + b(n-5)$ $(n \geq 5)$
  - $c(n)$ を n 円を1円玉と5円玉と10円玉で返す場合の数だとすると
    - $c(n) = b(n)$ $(n < 10)$
    - $c(n) = b(n) + c(n-10)$ $(n \geq 10)$

練習問題8−2

change.hs

```
import System.Environment

main = do args <- getArgs
          print $ change 500 $ read $ head args

change 500 n = if n < 500 then change 100 n else ...
change 100 n = ...
change 50 n = ...
...
change 1 n = 1
```

- change c n は c 円以下のコインを使って n 円のおつりを返す場合の数です.

```
% ghc change.hs
...
% ./change 30
16
% ./change 1000
248908
%
```

リストの再帰呼び出し

- リストは実は空リスト[]と(:)からできています.
  - `[] :: [a]`
  - `(:) :: a -> [a] -> [a]`
  - `[1, 2, 3] = 1:(2:(3:[]))`
  - `1:[2]` → `[1, 2]`
  - `5:[]` → `[5]`

- リストに対するパターンマッチでは[]の場合と(:)の場合を書きます.

```
map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = (f x) : (map f xs)
```

mapの再帰呼び出し

再帰呼び出し(例)

- リストの長さを返す関数**length**
  - `length [] = 0`
  - `length (x:xs) = 1 + length xs`
- リストの数の合計を返す関数**sum**
  - `sum [] = 0`
  - `sum (x:xs) = x + sum xs`
- 2つのリストをつなげる関数 **(++)**
  - `(++) :: [a] -> [a] -> [a]`
  - `(++) [] ys = ys`
  - `(++) (x:xs) ys = x : ((++) xs ys)`
- 2重のリストを結合して1重にする関数**concat**
  - `concat :: [[a]] -> [a]`
  - `concat [] = []`
  - `concat (x:xs) = x ++ concat xs`

練習問題8－3

reverse.hs

```
import System.Environment

main = do args <- getArgs
          print $ myreverse $ map read args

myreverse :: [Int] -> [Int]
myreverse []     = []
myreverse (x:xs) = ...
```

- リストを反転させる **myreverse** を書きなさい.
  - 空リストの反転は空リストです.
  - **(x:xs)** を反転させたいとき，**xs** を反転させてどうすれば良いでしょうか

```
% ghc reverse.hs
...
% ./reverse 1 2 3 4 5
[5,4,3,2,1]
%
```

練習問題8－4

sort.hs

```
import System.Environment

main = do args <- getArgs
          print $ mysort $ map read args

mysort :: [Int] -> [Int]
mysort []     = []
mysort (x:xs) = ...

myinsert :: Int -> [Int] -> [Int]
myinsert x []     = [x]
myinsert x (y:ys) = ...
```

- リストを小さい順に並び変える **mysort** を書きなさい.
  - 空リストの並び替えは空リストです.
  - **(x:xs)** を並び変えたいとき, **xs** を並び替えて, **x** を適切に挿入すればよいでしょう

```
% ghc sort.hs
...
% ./reverse 5 3 7 2
[2,3,5,7]
%
```

練習問題8－5

- 引数として与えられた数字をローマ数字として出力するroman.hsを書きなさい.

実行例

```
% ./roman 1111
MCXI
% ./roman 1954
MCMLIV
% ./roman 1990
MCMXC
% ./roman 2018
MMXVIII
```

roman.hs

```
import System.Environment

main = do args <- getArgs
          putStrLn $ roman $ read $ head args

roman::Int -> String
roman ....
```

- ローマ数字では7つの文字を組み合わせることで数字を表します.

文字	I	V	X	L	C	D	M
値	1	5	10	50	100	500	1000

- 個々の文字に対応した数字の合計になります.
- 大きな数から順番に書いていきます.
- 同じ文字が4つ続くのを防ぐために，引き算を使った書き方が用いられます.

文字列	IV	IX	XL	XC	CD	CM
値	4	9	40	90	400	900