関数型プログラミング
第2回　基本（1）関数とリスト

萩野　達也
hagino@sfc.keio.ac.jp

プログラム開発環境

- プログラム開発環境CUI vs GUI
  - CUI（Character User Interface）またはCLI（Command Line Interface）
    - 単純で軽い
    - コンパイラとライブラリを使ってプログラム開発
    - テキストエディタを使ってプログラムを書く
  - GUI（Graphical User Interface）
    - 現代的だが重い
    - エディタ，コンパイラ，デバッガなどが一体となっている
    - 例：eclipse, Xcode, Visual Studio

- CUI
  - UNIX (Linux): シェル（sh, csh, tcsh, bash）
  - Mac OS X: ターミナル
  - Windows: コマンドプロンプト

- テキストエディタ
  - UNIX (Linux): vi (vim), emacs
  - Mac OS X: TextEdit, mi, emacs
  - Windows: notepad, xyzzy
  - 非依存：atom

UNIXの基本コマンド

- CUIの基本
  - 実行するコマンドを入力する
  - コマンド名と引数を与える
  - current working directoryを正しく設定すること
  - folder = directory

% command arg1 arg2 arg3

引数

プロンプト

- シェルの基本コマンド

コマンド	意味
pwd	current working directoryを表示（print working directory）
cd *dir*	ディレクトリを*dir*に変更（change directory）
ls *dir*	*dir*にあるファイルの一覧を表示（list）
ls -l *dir*	*dir*にあるファイルの一覧を詳しく表示（long list）
cat *file*	*file*の中身を表示（concatenate）
more *file*	*file*の中身を1ページずつ表示
mkdir *dir*	新しいディレクトリ*dir*を作成（make directory）
rmdir *dir*	ディレクトリ*dir*を削除する（remove directory）
rm *file*	*file*を削除する（remove）
command < *file*	*command*の入力を*file*からにする（入力リダイレクション）
command > *file*	*command*の出力を*file*にする（出力リダイレクション）

削除したものを戻すことはできない
ゴミ箱に移すのとは異なる

ファイルの中身を表示する

- UNIXのcatコマンドに似たものをHaskellで書いてみましょう.
  - あたえられたファイルの中身を表示する.

cat.hs

```
main = do cs <- getContents
          putStr cs
```

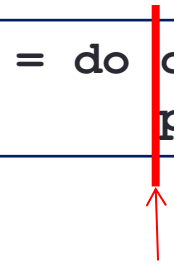
```
% ghc cat.hs
...
% ./cat < cat.hs
main = do cs <- getContents
          putStr cs
%
```

- "./"は現在のディレクトリを表す.
- "./cat"は現在のディレクトリの"cat"プログラムを意味する
- Windowsでは"./cat"のかわりに".¥cat.exe"としてください.

catプログラム

- getContents
  - アクション
  - アクションが実行されると標準入力の値を読み込む.
  - 標準入力から読み込まれるすべてを，一つの文字列として返す.
- putStr cs
  - 文字列csを標準出力に出力するアクション
- do式
  - 複数のアクションを上から順番に評価していく.
  - どこかでアクションが失敗すると，そこで停止する.
  - cs <- getContents
    - getContentsアクションを評価した結果の文字列を変数csに束縛する.
    - 変数csはそれ以降のアクションで参照することができる.

```
main = do cs <- getContents
          putStr cs
```



レイアウト構文

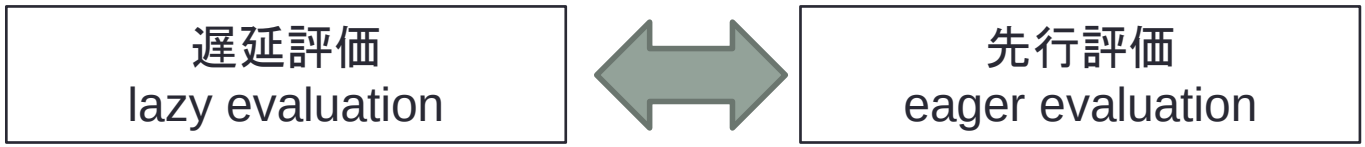
- 同じインデントの行がグループ化される
- ブロック構造
- {や}で囲む必要がない.

遅延評価（lazy evaluation）

cat.hs

```
main = do cs <- getContents
          putStr cs
```

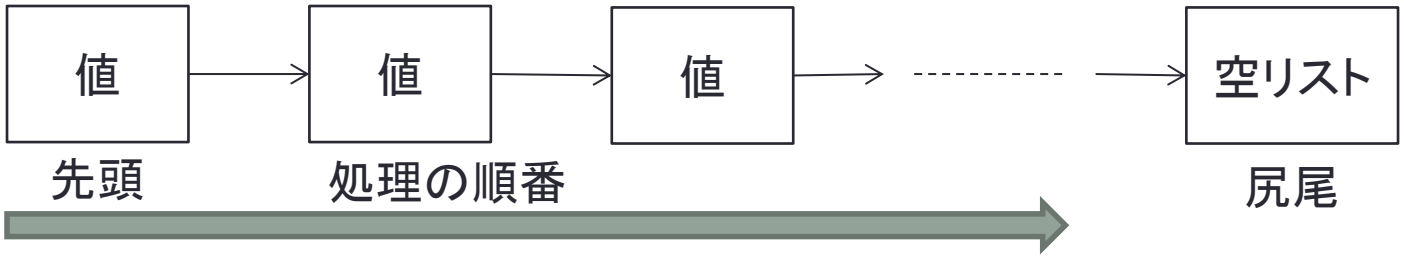
- getContentsは標準入力からの入力のすべてを一度に読み込むわけではない.
  - csは標準入力に入ってくる文字列を表している.
  - 実際のcsの中身は，参照されたときに初めて標準入力から読み込まれる.
- putStrはcsの中身をアクセスする. そのことにより実際の標準入力から読み込まれる.
  - putStrが必要とする文字数だけ読み込まれる.
  - 端末からの入力の場合には，行ごとに読み込まれる.



必要になった時にはじめて評価する
なるべく評価を遅らせる

先に評価してしまう
積極的に評価する

リスト



- 複数の値をつなげる.
- 先頭から順番に処理する.
- 最後の値は「空リスト」
  - C言語におけるヌルリストにあたる.

- リストは同じ型の値しか入れることができない.
  - 異なる方の値を混ぜることはできない(整数と文字とか).
- リスト内の値を変更することはできない.
  - 配列とは異なる.

リストリテラル

- [1, 2, 3]
  - 数字1と2と3のリスト
- ["aa", "bb", "cc"]
  - 3つの文字列のリスト
- ['a', 'b', 'c']
  - 3文字のリスト
  - "abc"と同じ
- 同じ型の値しか，同じリストには入れることができない
  - [1, 'c', "string"] はダメ
  - [1, [2, [3]]] はダメ
- []
  - 空リスト

ファイルの行数を数える

countline.hs

```
main = do cs <- getContents
          print $ length $ lines cs
```

- 上記のプログラムを実行

```
% ghc countline.hs
...
% ./countline < countline.hs
2
%
```

countlineの詳細

countline.hs

```
main = do cs <- getContents
          print $ length $ lines cs
```

- '$' 演算子
  - '+' や '*' とおなじ2項演算子
  - 'x $ y' の意味は 'x(y)'
  - 'length $ lines cs' は 'length(lines cs)'
  - 'print $ length $ lines cs' は
    - print(length(lines cs))
  - 'print length lines cs' は
    - (((print length) lines) cs)

countlineの詳細（つづき）

- 'lines' 関数
  - 文字列を行ごとに分ける
  - `lines "aaa¥nbbb¥nccc¥n"` → `["aaa", "bbb", "ccc"]`
  - `lines "aaa¥n"` → `["aaa"]`
  - `lines "aaa"` → `["aaa"]`
  - `lines "¥n"` → `[""]`
  - `lines ""` → `[]`

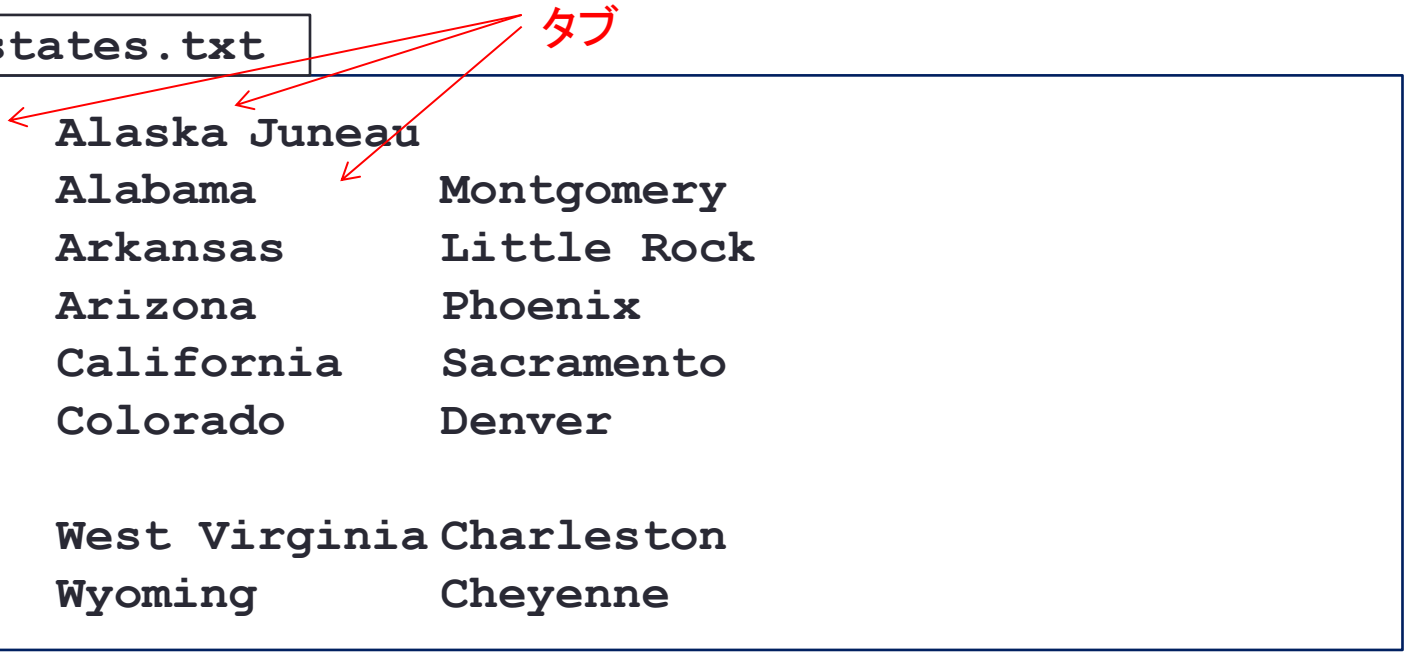
- 'length' 関数
  - リストにつながれた値の数を数える
  - `length [1, 2, 3, 4]` → `4`
  - `length [5, 11]` → `2`
  - `length []` → `0`
  - `length ["aa", "bb"]` → `2`
  - `length ["aa"]` → `1`
  - `length [""]` → `1`
  - `length "string"` → `6`
  - `length "str"` → `3`
  - `length ""` → `0`

- 'print' 関数
  - 値を出力するアクション
  - 値は文字列にシリアライズされる.

USA-states.txt

- アメリカの州名とその省略形と州都
  - See http://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States

USA-states.txt



- タブで区切られている
- 以下からダウンロード:

  `http://web.sfc.keio.ac.jp/~hagino/fp16/USA-states.txt`

ファイルの先頭10行を表示

head.hs

```
main = do cs <- getContents
          putStr $ firstNLines 10 cs

firstNLines n cs = unlines $ take n $ lines cs
```

- 上記プログラムを実行

```
% ghc head.hs
...
% ./head < USA-states.txt
AK      Alaska    Juneau
AL      Alabama   Montgomery
AR      Arkansas Little Rock
AZ      Arizona   Phoenix
CA      California        Sacramento
CO      Colorado Denver
CT      Connecticut       Hartford
DE      Delaware Dover
FL      Florida   Tallahassee
GA      Georgia   Atlanta
```

関数への引数の適用

func arg1 arg2 ····

- 関数に引数を適用する:
  - *func arg*
- 引数が2つある場合:
  - *func arg1 arg2*
- 引数が3つの場合:
  - *func arg1 arg2 arg3*
- 括弧は必要ありません:
  - *func arg1 arg2* → (*(func arg1) arg2*)
  - *func arg1 arg2 args* → (((*func arg1*) *arg2*) *arg3*)

func arg1 arg2 ≠ *func(arg1, arg2)*

関数の定義

func param1 param2 ・・・・ = body

- `firstNLines n cs = unlines $ take n $ lines cs`
  - '`firstNLines`' を定義
  - '`firstNLines`' は2つの引数 '`n`' と '`cs`' を受け取ります.
  - 引数は関数の本体で参照できます.
  - 本体は '`unlines $ take n $ lines cs`'

'unlines' と 'take'

- 'unlines' 関数
  - 'lines' 関数の逆.
  - リストの文字列を改行で区切りながらつなげる.
  - unlines ["aaa", "bbb", "ccc"] → "aaa¥nbbb¥nccc¥n"
  - unlines ["aaa"] → "aaa¥n"
  - unlines [""] → "¥n"
  - unlines [] → ""
  - unlines ["aaa¥n"] → ["aaa¥n¥n"]

- 'take n' 関数
  - リストの先頭から n 要素を取り出したリストを作る.
  - リストが n より短い時には, リストをそのまま返す.
  - take 3 [5, 2, 4, 6, 8] → [5, 2, 4]
  - take 3 [5] → [5]
  - take 3 [] → []
  - take 3 "string" → "str"
  - take 0 [1, 2, 3] → []

練習問題2-1

head.hs

```
main = do cs <- getContents
          putStr $ firstNLines 10 cs

firstNLines n cs = unlines $ take n $ lines cs
```

- 上のプログラムを '$' を使わないように書き直しなさい.

注意

宿題においては原則授業で習ったことしか利用してはいけません.
ネットで調べたもので授業でやっていないものは, 正解とはみなしません.
前もって知っていたとしても授業どおりの答えを書いてください.

'reverse' と 'words'

- 'reverse' 関数
  - リストの要素の順番を逆転させたリストを返す.
  - `reverse [1, 2, 3]` → `[3, 2, 1]`
  - `reverse []` → `[]`
  - `reverse "string"` → `"gnirts"`
  - `reverse ""` → `""`
  - `reverse ["abc", "def", "ghi"]` → `["ghi", "def", "abc"]`
- 'words' 関数
  - 文字列を単語に分割する.
  - 空白(タブ, 改行を含む)で単語は区切られているものとする.
  - `words "This is a pen."` → `["This", "is", "a", "pen."]`
  - `words " a(1, 2, 3) "` → `["a(1,", "2,", "3)"]`
  - `words "a¥nb¥nc¥n"` → `["a", "b", "c"]`
  - `words ""` → `[]`

練習問題2-2

reverse.hs

```
main = do cs <- getContents
          putStr $ reverseLines cs

reverseLines cs = ...
```

- ファイルの行を逆順に出力するプログラムを完成させなさい.

```
% ghc reverse.hs
...
% ./reverse < USA-states.txt
WY      Wyoming  Cheyenne
WV      West Virginia    Charleston
WI      Wisconsin        Madison
WA      Washington       Olympia
VT      Vermont  Montpelier
...
AK      Alaska   Juneau
```

練習問題2-3

tail.hs

```
main = do cs <- getContents
          putStr $ lastNLines 10 cs

lastNLines n cs = unlines $ takeLast n $ lines cs

takeLast n ss = ...
```

- ファイルの最後の10行を出力するプログラムを完成させなさい.

```
% ghc tail.hs
...
% ./tail < USA-states.txt
SD        South Dakota      Pierre
TN        Tennessee         Nashville
TX        Texas     Austin
UT        Utah      Salt Lake City
VA        Virginia  Richmond
VT        Vermont   Montpelier
WA        Washington        Olympia
WI        Wisconsin         Madison
WV        West Virginia     Charleston
WY        Wyoming   Cheyenne
```

練習問題2-4と2-5

countbyte.hs

```
main = do cs <- getContents
          print ...
```

- ファイルのバイト数を出力する.

countword.hs

```
main = do cs <- getContents
          print ...
```

- ファイルの単語数を出力する.

練習問題2-6

middle.hs

```
main = do cs <- getContents
          putStr $ takeNLines 10 5 cs

takeNLines n m cs = ...
```

- ファイルの先頭の10行を飛ばして，その次からの5行を出力するプログラムを完成させなさい．

```
C:¥> ghc middle.hs
...
C:¥> middle < USA-states.txt
HI       Hawaii   Honolulu
IA       Iowa     Des Moines
ID       Idaho    Boise
IL       Illinois Springfield
IN       Indiana  Indianapolis
```

関数とアクションのまとめ

関数	例	意味
`putStr`	`putStr cs`	**文字列csを出力するアクションを返す**
`putStrLn`	`putStrLn cs`	**文字列csを出力し，改行を出力するアクションを返す．**
`print`	`print x`	xの値を出力するアクションを返す．
`length`	`length xs`	リストxsの長さを返す．
`take`	`take n xs`	リストxsの先頭からn要素だけのリストを返す．
`reverse`	`reverse xs`	リストxsを逆順に並び替えたリストを返す．
`lines`	`lines cs`	文字列csを行ごとに分割したリストを返す．
`unlines`	`unlines xs`	リストxsの文字列を改行を挟んでつなげた文字列を返す．
`words`	`words cs`	文字列csを単語のリストに分割する．

アクション	例	意味
`getContents`	`getContents`	An action of reading the standard input