プログラミング言語論
第９回

関数型プログラミング

Haskell入門

命令型・宣言型プログラミング

- 命令型プログラミング
  - 「問題を解く手順を記述する」ことでプログラムするパラダイム
    - 手続き型プログラミング
    - 構造化プログラミング
    - オブジェクト指向プログラミング
- 宣言型プログラミング
  - 「問題の性質を記述する」ことでプログラムするパラダイム
    - 関数型プログラミング
    - 論理型プログラミング
    - データベース言語

関数型プログラミング言語とは（1/3）

- **数学の関数**のように，関数を扱う言語
  一方，「プログラミング言語の関数は，数学で教わった関数とは，意味が違ったな」という記憶を多くの人が持っている．
- 複数の式を関数の適用のよって組み合わせていくプログラミング形式
- 基本的には全部が式

だからどうなる？

関数型プログラミング言語とは（2/3）

一度宣言・定義した変数，配列の中身を絶対に変更しないようにプログラミングする方法．データははじめから最後まで永続する参照透過性を持つ．

- 代入出来ない
  - 全部const宣言．宣言したら全て定数
- For文が使えない
  - ありとあらゆるfor文は再帰呼び出しで書ける
- 関数に関数を引数として渡す
- 永続データプログラミング

関数型プログラミング言語とは（3/3）

- 極論をいうと,

「変数への代入と制御構造を全部排除して書け！」

どうやって！？

- for文はだいたい再帰で書ける
- Map-reduce演算

これまでのプログラミングとは全く違うアイディアでプログラムを書くことになる→手続き型からの**頭の切り替えが必要**

手続き型と関数型（宣言型）で全て同じ機能を実現可能なことは証明済み→詳細は**ラムダ計算**

手続き型プログラミングの問題

- int a=10 (変数の定義)
- a=20　（変数への代入）

- この代入がいつどこで行われるかわからない問題
  - 構造化言語における変数
  - 知らないうちに（忘れているうちに）オブジェクトの状態が変わっている
  - 複数のオブジェクトが絡み合ってるときに起こる「想定外の状態」になる可能性
  - オブジェクトが状態を持つのが悪い？→メンバ変数への代入が禁止されていたら、「オブジェクトは状態を持たない」
- つまり，「**代入**」が諸悪の根源？

関数型での解決

- （純粋）関数型言語では，
  - 代入を禁止してしまえばそもそもこの問題が起きないのでは？
    - 代入を一切禁止してしまう
- 一度宣言・定義したら代入禁止
  - その結果，a++などもできない
  - for loopも書けなくなる
- これでプログラムって書けるの？？？
  - 書けるように書くのが関数型プログラミング
  - ありとあらゆるプログラムが代入しなくてもかける
  - ラムダ計算が基本.

メジャーな関数型言語一覧

- 静的型付き：Clean，F#，Haskell，OCaml，Scala，SML
- 動的型付き：Clojure，Erlang，（Lisp）
  - コンパイル時に型の検査をするのが静的，実行時が動的

  ※Common Lisp, Emacs Lispは関数型というには微妙．代入を前提にして作られている．

  ※Rustは，関数型の機能を沢山取り入れた手続き型言語

全部が関数として定義した言語Lazy K


```
K
(SII(S(K(S(S(K(SII(S(S(KS)(S(K(S(KS)))(S(K(S(S(KS)(SS(S(S(KS)K))(KK)))))
(S(S(KS)(S(KK)(S(KS)(S(S(KS)(S(KK)(S(KS)(S(S(KS)(S(KK)(SII)))
(K(SI(KK)))))))(K(S(K(S(S(KS)(S(K(SI))(S(KK)(S(K(S(S(KS)K)(S(S(KS)K)I)
(S(SII)I(S(S(KS)K)I)(S(S(KS)K)))))(SI(K(KI)))))))))(S(KK)K)))))))(K(S(KK)
(S(SI(K(S(S(S(S(SSK(SI(K(KI))))(K(S(S(KS)K)I(S(S(KS)K)(S(S(KS)K)I))
(S(K(S(SI(K(KI)))))K)(KK))))(KK))(S(S(KS)(S(K(SI))(S(KK)(S(K(S(S(KS)K)))
(SI(KK)))))(K(K(KI)))))(S(S(KS)(S(K(SI))(SS(SI)(KK))))(S(KK)
(S(K(S(S(KS)K)))(SI(K(KI)))))))))(K(K(KI)))))))))(K(KI)))))(SI(KK)))))
(S(K(S(K(S(K(S(SI(K(S(K(S(S(KS)K)I))(S(SII)I(S(S(KS)K)I)))))))K))))
(S(S(KS)(S(KK)(SII)))(K(SI(K(KI)))))))(SII(S(K(S(S(KS)(S(K(S(S(SI(KK))
(KI))))(SS(S(S(KS)(S(KK)(S(KS)(S(K(SI))K)))))(KK))))))(S(S(KS)
(S(K(S(KS)))(S(K(S(KK)))(S(S(KS)(S(KK)(SII)))(K(S(S(KS)K)))))))(K(S(S(KS)
(S(K(S(S(SI(KK))(KI))))(S(KK)(S(K(SII(S(K(S(S(KS)(S(K(S(K(S(S(KS)(S(KK)
(S(KS)(S(K(SI))K))))(KK)))))(S(S(KS)(S(KK)(S(K(SI(KK)))(SI(KK)))))
(K(SI(KK)))))))(S(S(KS)(S(K(S(KS)))(S(K(S(KK)))(S(S(KS)(S(KK)(SII)))
(K(SI(K(KI)))))))(K(K(SI(K(KI)))))))(S(K(SII))(S(K(S(K(SI(K(KI))))))
(S(S(KS)(S(KK)(SI(K(S(K(S(SI(K(KI)))))K)))))(K(S(K(S(SI(KK))))
(S(KK)(SII)))))))))(K(SI(K(KI)))))))(S(S(KS)K)I)
(SII(S(K(S(K(S(SI(K(KI)))))K))(SII)))))
```

Lazy Kは組み込み関数が3つしかない純粋関数型言語

左の例は，素数を計算するプログラム

関数型言語のプログラム例

- 例題： ０～１０までの総和を計算する問題を解く
  - 左：手続き型，右：関数型．漸化式を定義して使う．

```
total = 0;
for i = 0 to 10 do
  total = total + i;
done;
```

Fortran

```
let
    sum x = if x == 0 then  0
            else  x + sum (x - 1)
in
sum 10
```

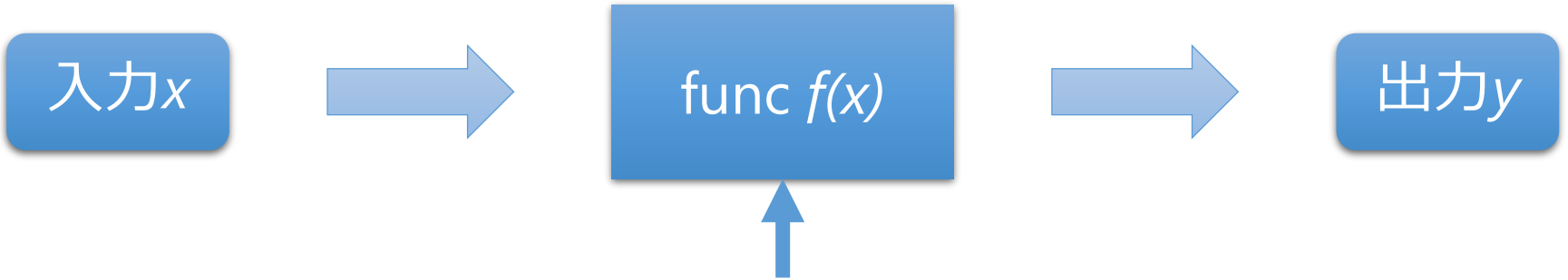
Lisp

- 関数型は，宣言型のプログラミング言語の一種
  - 対義語：手続き型（これまでの言語）

関数型と手続き型の関数の違い

数学の関数は入力xに対して，出力yは常に同じ値．これを，**参照透過性**と呼ぶ．もしくは，**副作用**が無いという．

手続き型プログラムの関数は入力xに対して，出力yは常に同じ値とは限らない．これは，グローバル変数のせい



関数型言語の特徴

- 第一級関数
  - 関数を数値などの型と同様に取り扱いできること.
- 高階関数
  - 関数を返したり，引数として受け取ったりできる関数のこと.
- 純粋関数
  - 状態を持たない関数のこと（数学の意味の関数）.
- 不変状態
  - 状態を全く変更できないこと（const）

ラムダ計算λ（lambda）-calculusとは

- 論理学者（今から見れば、理論計算機科学者）の**A. チャーチ**が考案した計算モデル
- 計算について理論的に考えるときに使う標準体系の一つ（チューリング機械もその１つ）
- 関数型言語の理論的基礎
- 関数の合成と適用によって計算を表現
- 関数と対象を区別しない

λを使った関数の表現

- 通常用いる関数の表現：

$$f(x) = x^2 + x + 1$$

  - 引数を $f(x)$ などと，括弧付きで表す．
  - 関数の名称 f などでつけて表す．名称が必要．
  - 関数を適用する場合はこの名称を用いて、$f(4)$などとする．
- λ計算では、上の関数を次のように書く．

$$\lambda x\,.\,x^2 + x + 1$$

あるいは、型付のλ計算では次のように書く．

$$\lambda x \in \mathbb{N}\,.\,x^2 + x + 1$$

λを使った関数の表現

$$\lambda x.\ x^2 + x + 1$$

- 関数名を付けずに、関数を表現できる。
- そのまま適用可能

$$((\lambda x.\ x^2 + x + 1)\ 4) = 4^2 + 4 + 1 = 21$$

- 名前をつけることも可能

$$f \equiv \quad \lambda x.\ x^2 + x + 1$$

$$(f\ 4) = 21$$

- 名前なしの関数を，無名関数という（LISPの用語）

関数型言語の歴史

1930年代：λ算法（λ計算）の登場
1958年：Lisp→λ算法に影響を受けて開発
1973年：ML（Meta-Language）
1975年：Scheme（LISPの一種）
1984年：Common Lisp
1986年：Erlang：並列処理指向のプログラミング言語
1990年：Haskell
1996年：OCaml　オブジェクト指向型のML実装
2003年：Scala（JVM上，twitterのバックエンドシステム）
2005年：F#　マイクロソフトの言語．Ocamlからの発展
2007年：Clojure　Lispの方言（JVM上）

関数型言語が流行る兆し

- ラムダ式：最近あらゆる言語に取り入られ始めた関数型言語の仕組み
- スクリプト言語よりも速く，Javaくらいの速度が出るようになった
- バグが入りづらい言語仕様
- 並列プログラミングに向いている
- 普段使わない書き方を覚えることで，プログラミングの考え方の幅を広げる
- コードを書く量が少なくていい

Haskell（はすける）

- 純粋関数型の強い静的型付け言語
- 数学者ハスケル・カリーに由来
- 設計者　サイモン・ペイトン・ジョーンズ

Haskellの特徴

- プログラムが短くてすむ（開発の高速化）
- 強力な静的型チェック：コンパイルが通れば、実行時にプログラムがクラッシュすることは（ほとんど）ない
- 強力な型推論：型を書くのサボれる（楽）
- 参照透過性を持たない部分を局所化できる（C++のイメージでいうとプログラムのほとんどの部分がconst)

クイックソート in Haskell


```
qsort []     = []
qsort (p:xs) = qsort lt ++ [p] ++ qsort gteq
               where
                 lt   = [x | x <- xs, x < p]
                 gteq = [x | x <- xs, x >= p]
```

その他の関数型言語Haskellの機能

- 高階関数
- カリー化
- パターンマッチング
- 遅延評価
- モナド
- 型推論
- map-reduce

関数型の関数型らしさを理解するには
基本的な文法の体感がまず必要

GHC

- The Glasgow Haskell Compiler
  - https://www.haskell.org/ghc/
  - GNUじゃない
- 広く使われているHaskellコンパイラ
- 自分の手元でHaskellが利用できるように環境構築してみてください．GHCを利用するのがおすすめ．どうしても難しい人はHaskell.org などのウェブ上の動作環境を利用してください．
- CSEでも使えるように設定してあります．

小テスト：Haskell入門１

Hello Worldを実行する

1. hello.hsというファイルを作成し，下記1行を書く
2. main = putStrLn "Hello World!“
3. ghc hello.hs を実行すると実行ファイルa.outができる
4. ./a.out
5. Hello World!

ghc ○○.hs
　で実行ファイルa.outが生成（コンパイラの設定による）
ghc ○○.hs –o ××
　で実行ファイル名××のファイルが生成

小テスト：Haskell入門２

Hello Worldを実行する

1. hello.hsというファイルを作成し，下記1行を書く
2. main = putStrLn "Hello World!“
3. runghc hello.hs を実行するとスクリプト言語っぽく実行される(実際にはウラでコンパイルしてる)
4. Hello World!

小テスト：Haskell入門２

インタラクティブに計算する

ghciとタイプ
1+1
＞２
:qで終了
（:？でヘルプ）

レポート9-1

- 下記計算をインタラクティブシェルで実行し，計算結果を確かめよ．また，その実行結果をレポートにまとめよ．
  - 2+3*4
  - (2+3)*4
  - sqrt(3^2 + 4^2)
- 下記を打ち込み，挙動を確認せよ
  - reverse “cse”
  - length [1,2,3,4]
  - succ 8
  - min 5, 6
  - max 100.5, 200.2
  - div 92 10
  - 92 `div` 10

レポート9-2

- 次のプログラムを打ち込み，コンパイルして動作を確認せよ．動作例をレポートにまとめよ．
  - 階乗計算をforループ無しで行った例

```
fact 0 = 1
fact n = n * fact(n-1)


main = do
    print $ fact 5
```

その他余裕がある人に

- 次のサイトの記述をみて基本ルール，文法を予習(（入力・実行）せよ.
  - http://qiita.com/7shi/items/145f1234f8ec2af923ef

(Haskell超入門@Qiita）

- 注意事項
  - Haskellは改行やタブ（空白）などの並び順に意味があります．タブなどの位置をそろえているサンプルコードはそのまま書きましょう.

言語学習における写経について

最強のプログラミング勉強法が写経である理由

https://wirelesswire.jp/2018/06/65757/

言語学習のコツ

- チュートリアルを読むだけでなく、出てくる例文を打ち込み、動作を確認する
- 少し変えてみる
- 小さいコードを書く

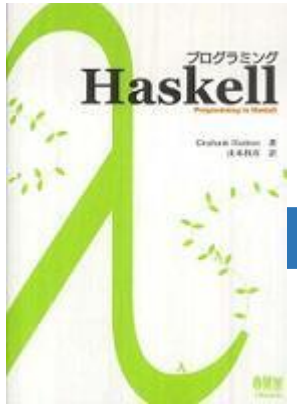
エディタを拡張している人へ

- emacs には，Haskell-modeが入れられます
  - https://github.com/haskell/haskell-mode
  - 英語で書いてるチュートリアルどおりにやれば入ります
- VS code (visual studio code)やAtom，Sublime Textの人は自分で探してみて下さい.

ウェブ上の資料

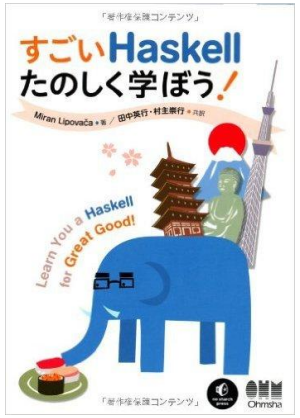
- Haskell の文法の特徴を把握したいときには次の記事がよくまとまっています．
  https://qiita.com/gomi_ningen/items/581eaed9cf232cac423a
- （Haskell基本構文まとめ@Qiita)
- https://wiki.haskell.org/10分で学ぶHaskell

Haskellの参考書



プログラミングHaskell：定番．**入門書ではない．**

- 説明スライド
  - 原著
    - http://www.cs.nott.ac.uk/~pszgmh/book.html
  - 日本語
    - http://www.ist.aichi-pu.ac.jp/lab/yamamoto/program_languages/2013/index.html



すごいHaskellたのしく学ぼう：この本がお薦め．

巷でもとても評判がよい．

練習問題

- 時間がある人に
  - Haskell.org のチュートリアルをやってみる