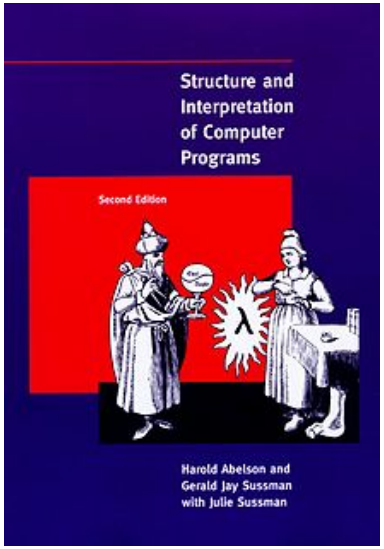
プログラミング言語 No.2-1

2025.4.11 中川博之

関数型言語

関数型言語とは

- ラムダ計算 (Lambda Calculus)をベースとしたプログラミング言語
- ラムダ計算は「関数」の評価(evaluation)と適用(application)で計算をモデル化した計算体系
- LISP (List Processor) ：もともとはリストの処理のための言語
  小出誠二，武田英明：人工知能用言語Lispの今と将来，人工知能学会誌 Vol.24, No.5, pp. 681-690, 2009.
  https://www.jstage.jst.go.jp/article/jjsai/24/5/24_681/_pdf
- LISP には多くの方言 (派生した言語) がある



Lisp のプログラム例

Lisp で書いた階乗のプログラム

```
(defun factorial (n)
    (if (n = 0) 1
     (* n (factorial (- n 1)))))
```

Lisp は関数の始まりと終わりを(と) で表すため，プログラムは括弧だらけになる.

Common Lisp で書いたFizzBuzzのプログラム

```
(defun fizzbuzz(i)
  (cond
    ((= 0 (mod i 15)) “FizzBuzz")
    ((= 0 (mod i  3)) "Fizz")
    ((= 0 (mod i  5)) "Buzz")
    (t (write-to-string i))))
```

Common Lisp は Lisp の方言の中で最も有名なものの一つ.

関数とは

「関数」という言葉には2種類の意味がある

- 数学的な意味での関数
  - 定義域と値域との関係を表したもの
  - 定義域に含まれる値を入れたら結果が必ず一つ返る
  - 入力が同じなら，返ってくる値はいつでも同じ
- プログラム的な意味での関数
  - 必ずしもある入力に対していつも同じ結果が返るとは限らない（例：time関数).
  - 意味的にまとまった一つの作業に名前をつけたもの．返り値付きのサブルーチン

関数型言語は数学的な意味での関数がベース

- つまり，引数と結果との関係を書く
- すべてを式と考える（ラムダ計算）
  - 関数も式
  - 数字も式
- 利点は
  - 副作用がない (純粋型)，もしくは少ない (非純粋型)
  - プログラムを数学的に検証可能
  - 高階関数が書きやすい

副作用 (side effect)

- 関数型言語では，式の値を計算して求めることを「評価(evaluation)」という
- 副作用は評価以外の動作
  例：変数への値の代入
- 副作用がないと参照透過性(Referential Transparency) が保証される
  - 参照透過性: 同じ引数で呼び出した関数は常に同じ結果を返す

  すべての関数で参照透過性が保証されている関数型言語を純粋型の関数型言語という.

関数型言語では変数に値を代入することは基本的にはできない

関数型言語の基本的な特徴

- 変数への代入ができない
- ループ文 (C のwhile文やfor分のようなもの）がない
  → ループは関数の再帰呼び出しで実現
- 強力な型システムを持っているものが多い
  → 処理系が型を推論

型システム

- プログラミング言語の型システムのベースとなっている型理論は数理論理学の一つ
- 型のチェックをすることでプログラムのエラーを発見することが可能
- 関数型言語は特に型システムと相性がよい
  （型システムは関数型であることとは直接の関係はないが，関数型言語のベースであるラムダ計算には型付ラムダ計算という体系が確立されている）
- 強い型付けがされている関数型言語では型を自動的に推論する型推論機能がある
  （他のプログラミング言語でも型推論機能を持っているものはある）

主な関数型言語

- Lisp
  非純粋，動的型付け，多数の方言がある
- Haskell
  純粋，強い静的型付け，型推論を行う
- ML
  非純粋，強い静的型付け，型推論を行う

本講義では，これ以降，MLを用いた演習を行なって関数型言語の考え方を身につける

Standard ML

ML (Meta Language)

- 1973年に Robin Milner らにより設計開発された関数型言語
- いくつかの処理系／方言がある
- Standard ML: MLの標準
- 実用的に使われてはいないが，関数型言語の特徴を勉強するのによい
- 処理系
  - Standard ML of New Jersey (SML/NJ)
    - https://smlnj.org/ ← 本講義で使用する処理系
  - SML#
    - https://smlsharp.github.io/ja/

SML / NJ

- インタプリタ
  - コンパイルして実行可能形式にするのではなく，インタプリタ内で解釈し実行
  - コンパイルする言語に比べると実行時間が遅い
  - プログラムを書いたらすぐに実行できることが利点

- 起動： > sml
- 終了： ctrl-d

基本事項

- ML は関数型言語.
  - ML のプログラムは「式」であり，その「式」の値を計算することを式を「評価する」と言う．ML は式を評価することでプログラムを実行
- ; が式の最後を示す．; が来たらそこまでの式を評価する.
- use "filename";
  - エディタなどで作成したファイル filename を読み込む.
- val <変数名> = <値> で変数を宣言できる.
  - プロンプト上で宣言すれば，その処理系の実行を終えるまで使える変数を宣言できる．let 内で用いれば局所変数の定義とみなされる

関数

- 関数名，パラメータ（引数）の順に記述.　　例) size "abc";
  - 3 + 5; のようにパラメータ間に関数名がくるものもある（中間演算子）
  - 算術記号など，慣習的なものは中間演算子として定義されている
  - 自身で定義した関数を中間演算子とすることもできる
- op ：中間演算子を関数名とするもの．例えば + は中間演算子なので，通常は
  - 5 + 3;　のように使うが，op を使うと
  - op + (5, 3);　のように引数を演算子より後ろに書くことができる
- 型の表示：例）val substring : string * int * int -> string
  - 関数 substring は string 型，int 型，int 型の3つのパラメータをとり，string 型の結果を返す

基本データ型

- 整数 (int)：0, 1234, 111111 など一つ以上の数からなる文字列．負の整数は単項の負記号 ~ (チルダ)で表す．例えば，~1
- 実数 (real)：0.0, 1.1 など .(ピリオド)を含む数からなる文字列．負の整数は単項の負記号 ~ (チルダ)で表す．例えば，~1.0
  - 注：0 は int, 0.0 は real.
- ブール値 (bool)：true または false
- 文字列 (string)：“foo” や “R2D2” のように引用符で囲まれた文字の並び．特殊文字列はC言語と同様に表現
- 文字 (char)：#"c" のように書くと char になる
  - 注："c" は string, #"c" が char

四則演算

強い型付け

同じ式の中で整数と実数を混ぜることはできない

- 整数
  - 2 + 3;
  - 4 – 10;
  - 3 * 4;
  - 9 div 3;
- 実数
  - 3.0 + 5.5;
  - 12.4 – 1.2;
  - –3.3 * 4.0;
  - –10.0 / 2.0;

実行例

```
$ sml
Standard ML of New Jersey v110.79 [built: Sat Oct 26
12:27:04 2019]
- 2 +3;
val it = 5 : int
- 4  - 10;
val it = ~6 : int
- 3 * 4;
val it = 12 : int
- 9 div 3;
val it = 3 : int
- 3.0 + 5.5;
val it = 8.5 : real
- 12.4 -1.2;
val it = 11.2 : real
- ~3.3*4.0;
- ~10.0/2.0;
val it = ~5.0 : real
- ~10/2.0;
```

注意

- 負の数は ~ で表現
- 強い型付け
  - 同じ式の中で整数と実数を混ぜることはできない
- MLでもLISPなどと同じく四則演算は関数として定義されているが，LISPとは違い，中置記法となっている
  - 関数は通常，関数名　引数の順番で書くが，中置記法では関数名を二つの引数の間に書く．

```
- -3.3 * 4.0;
stdIn:38.1 Error: expression or pattern begins with infix
identifier "-"
stdIn:38.1-38.11 Error: operator and operand don't agree
[tycon mismatch]
  operator domain: [- ty] * [- ty]
  operand:         real
  in expression:
    - 3.3
- ~10.0/2.0;
val it = ~5.0 : real
- ~10/2.0;
stdIn:40.1-40.8 Error: operator and operand don't agree
[overload conflict]
  operator domain: real * real
  operand:         [int ty] * real
  in expression:
    ~10 / 2.0
```

ML の基本型

- 整数型　　int
- 実数型　　real
- ブール型　　bool　　true / false
- 文字型　　char
- 文字列型　　string

MLにおいて，文字型と文字列型は別の型

宣言

- 関数宣言 fun <識別子> <パラメータリスト> = <式> ;
  例：定義
  - fun f x = x * 2;
    評価
  - f 3;

- 変数宣言 val <識別子> = <式> ;
  例： - val a = 2;

手続き型と関数型の違い

手続き型の主要な機能である以下の三つについて，関数型ではその捉え方が違う.

- 代入
- 条件分岐 (if文)
- 繰り返し (ループ)

宣言

- 関数宣言 fun <識別子> <パラメータリスト> = <式> ;
  例：定義
  - fun f x = x * 2;
    評価
  - f 3;

- 変数宣言 val <識別子> = <式> ;
  例： - val a = 2;

これを「代入」とは考えない． 束縛(bind)と呼ぶ.
valによって毎回新たな宣言が行われる.
同じ名前の変数をvalをすると,上書きされるのではなく新たにbindされる.

注意：一旦宣言した変数に,
- a = 3;
としても代入はできない！
(この式はaと3が等しいかどうかを比べる式となる．)

関数型言語では変数の宣言と代入は違う

(a をval していない状態で)
-fun f b = a * b;
とすると，a が宣言されていないというエラーとなる.

-val a = 2;
-fun f b = a * b;
とするとfは正常に定義される.
例えば以下のように実行すると,
-f 4;
の値は 8 になる.
ここで
-val a = 3;
とすると,
-a;
の値は 3 になるが,
-f 4;
の値は 12 ではなく 8 のままである.

MLでは変数に値を設定 (束縛) できるのは val だけ.
val a = 2;
で a は 2 にbindされる.

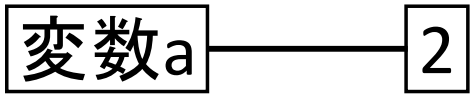
2回目の
val a = 3;
でこれまで 2 にbind されていた変数 a は新たに 3 にbindされる.

3 にbindされたaを参照したらその値は3だが，関数fを定義したときのaは2にbindされていたから，関数fではそのまま2 * b が評価される.

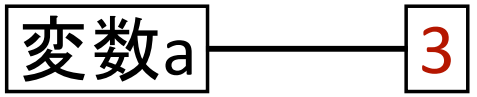
代入と束縛

代入

int a = 2;

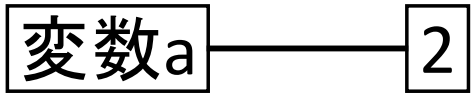


a = 3;

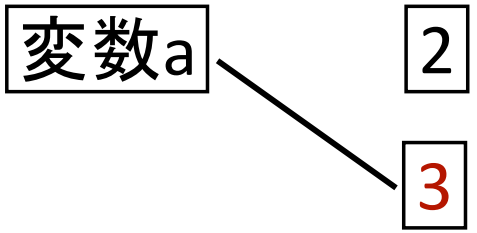


束縛

val a = 2;



val a = 3;



if 文

- 条件によって，実際に計算機が行うことを変えるという意味では手続き型も関数型も同じ.
- 手続き型のif文：制御フローの分岐.
- 関数型のif文：関数の一種．条件によって評価する式を変える.

関数型言語に「制御フロー」という考え方は存在しない.
どの式を評価するか.

if 文

- 条件によって，実際に計算機が行うことを変えるという意味では手続き型も関数型も同じ.
- 手続き型のif文：制御フローの分岐.
- 関数型のif文：関数の一種．条件によって評価する式を変える.

関数型言語の if 文は関数だから，必ず else を伴う.
else のない if 文は存在しない.

関数型言語に「制御フロー」という考え方は存在しない.
どの式を評価するか.

if 文

- 条件によって，実際に計算機が行うことを変えるという意味では手続き型も関数型も同じ.
- 手続き型のif文：制御フローの分岐.
- 関数型のif文：関数の一種．条件によって評価する式を変える.

関数型言語の if 文は関数だから，必ず else を伴う.
else のない if 文は存在しない.

関数型言語に「制御フロー」という考え方は存在しない.
どの式を評価するか.

関数なので定義域のすべての値に対して何かしらの評価結果を必ず返す.

ML の if 文

if <条件式> then <式> else <式> ;

引数 x が 0 ならば 1 をそうでないならば 0 を値とする関数 invert

```
fun invert x = if x = 0 then 1 else 0;
```

(MLプログラムでは改行は空白と同じ．どこで改行してもOK.)

	ML
等しい	=
小なり	<
以下	<=
大なり	>
以上	>=
等しくない	<>

関数型言語での繰り返し（ループ）

- 関数型言語では while文やfor文に相当するものはない
- ループは再帰を使って行う

階乗を計算するプログラム

C

```
#include <stdio.h>
#include <stdlib.h>

int
main(int agrc, char** argv){
  int n, fact;

  n = atoi(argv[1]);

  fact = 1;
  for(int i=1; i<=n; i++){
    fact = fact * i;
  }

  printf("%d\n", fact);
  return 0;
}
```

ML

```
fun fact n =
if n = 1 then 1
else n * fact (n-1);
```

関数型言語での繰り返し（ループ）

- 関数型言語では while文やfor文に相当するものはない.
- ループは再帰を使って行う.

階乗を計算するプログラム

C

```
#include <stdio.h>
#include <stdlib.h>

int
main(int agrc, char** argv){
  int n, fact;

  n = atoi(argv[1]);

  fact = 1;
  for(int i=1; i<=n; i++){
    fact = fact * i;
  }

  printf("%d¥n", fact);
  return 0;
}
```

ML

```
fun fact n =
if n = 1 then 1
else n * fact (n-1);
```

Cでも再帰で書くことはできるが，再帰よりループの方が実行速度が速いので，Cだとループで書くのが普通.

関数型言語は再帰を多用するので，一般的に実行速度は遅い.

ML で書いた FizzBuzz の例

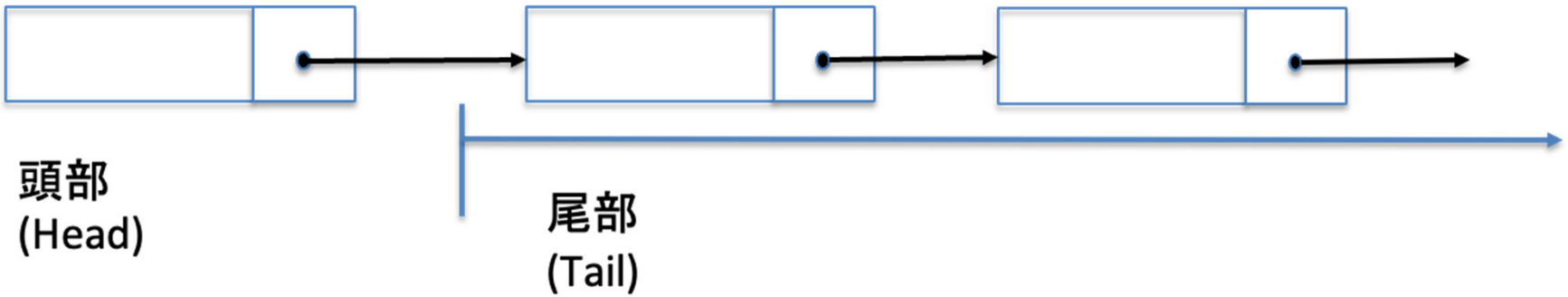
```
fun fizzbuzzNum n =
if ((n mod 3) = 0) andalso ((n mod 5) = 0) then "FizzBuzz"
else if (n mod 3) = 0 then "Fizz"
else if (n mod 5) = 0 then "Buzz"
else Int.toString n;

fun fizzbuzz (n, lst) =
if n = 1 then (Int.toString n)::lst
else fizzbuzz (n-1, (fizzbuzzNum n)::lst);
```

fizzbuzzNum は一つの数字のFizzBuzzの文字列を値とする関数

ここでは，引数で与えられたnまでのFizzBuzzをlstというリストに入れるようにしている

リスト

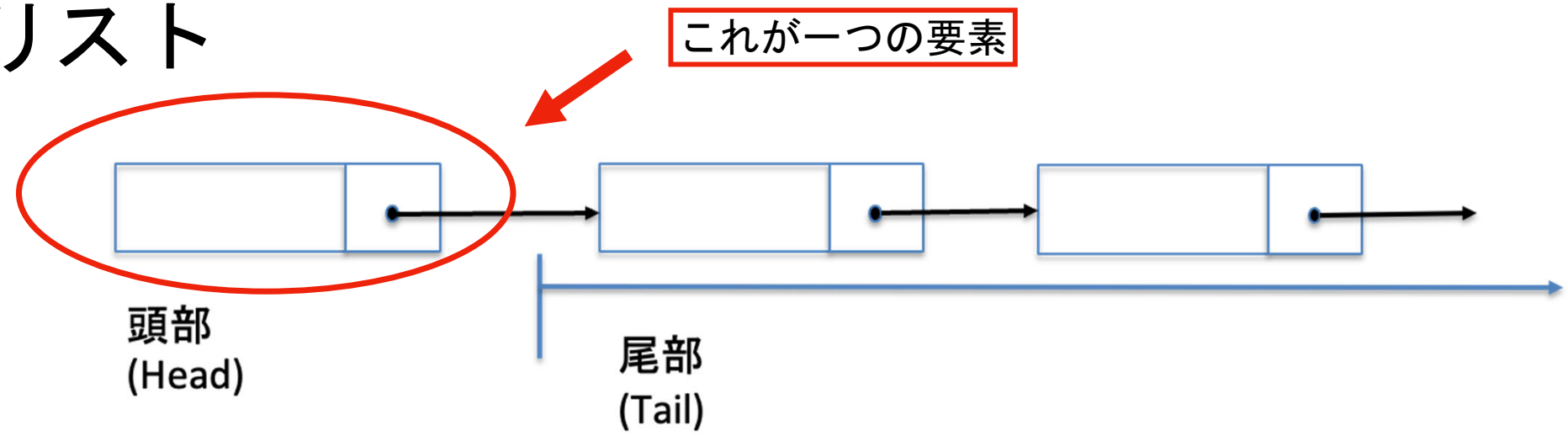


- 同じ型の複数のデータをひとまとまりにして扱える型
- 関数型言語には配列はない．　リストを使う．
- リストは長さは決まってない．可変．

リストの形式的定義

1. 空リストはリスト
2. リストの頭に要素を加えたものはリスト

リスト



- 同じ型の複数のデータをひとまとまりにして扱える型
- 関数型言語には配列はない．　リストを使う．
- リストは長さは決まってない．可変．

リストの形式的定義

1. 空リストはリスト
2. リストの頭に要素を加えたものはリスト

空リスト：要素が0個のリスト

リストは再帰的に定義されるので
関数型言語で扱いやすい

MLのリストの表記

[]で囲まれたものがリスト　要素の区切りは,

例：[1,2,3]
[3.4, 1.1, 10.6]
[“abc”, “defg”, “hi”, “jklmn”]
[[1,2,3], [5], []]　←　これは整数のリストのリスト

リストの要素は式であれば何でもOK.
ただし一つのリストの中のすべての要素の型は同じである必要がある.

リストに関する代表的な関数

- cons (コンス) 演算子 ::
  リストの先頭に新しい要素を追加する.
  例： 1::[2,3];
  結果が[1,2,3]になる
  例2： 1::2::3::nil;
  結果が[1,2,3]になる

リストの先頭に要素を追加する操作をLispでcons (construction の略らしい) と名づけて以来，この操作をする演算子をコンス演算子と呼ぶ.

- 連結演算子 @
  リスト同士をつなげて新しいリストを作る
  例3： [1,2,3]@[4,5,6];
  結果が[1,2,3,4,5,6]になる

リストの連結は，効率が悪いので，関数型言語ではあまり使われない.
リストを連結するときもコンス演算子を用いるのが好まれる.

参考文献

- A Gentle Introduction to ML
  https://www.cs.nmsu.edu/~rth/cs/cs471/sml.html
- その他チュートリアル
  https://www.smlnj.org/doc/literature.html
- Standard ML of New Jersey User's Guide
  https://www.smlnj.org/doc/
- 大堀淳著「プログラミング言語 Standard ML 入門 改訂版」共立出版株式会社, 2021
- Jeffrey D. Ulman 著，神林靖訳「プログラミング言語 ML」アスキー出版局, 1996
- Jeffrey D. Ulman, "Elements of ML programming ML97 edition", Prentice Hall, 1998