アルゴリズムとデータ構造入門
1.手続きによる抽象の構築
1.2 手続きとその生成するプロセス

奥 乃 博

大学院情報学研究科知能情報学専攻
知能メディア講座 音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/10/IntroAlgDs/
okuno@i.kyoto-u.ac.jp, okuno@nue.org

TAの居室は総合研究7号館4階418号室奥乃研
- 池宮 由楽(M1) 奥乃研・音楽情報処理G
- 坂東 宜昭(M1) 奥乃研・ロボット聴覚G
- 古川 孝太郎(M1) 奥乃研・ロボット聴覚G

NoStudent Left Behind

優秀レポート提出者

第1回
- 鈴木善樹
- 枡井啓貴
- 山田淳二
- 奥野僚介
- 小池拓矢
- 嶋 隼輝
- 秋田大空
- 磯西市路

第2回
- 中井 裕介
- **枡井啓貴**
- **山田淳二**
- 大家理和
- 小池拓矢
- 勝見久央
- 柴田健太郎
- 立松美雪

2

LaTeX でレポートを書く

```
\documentclass[a4paper,12pt]{article}
\usepackage{listings}
\begin{document}
\section{階乗の定義}
\begin{equation}
  n! =
    \left\{
      \begin{array}{ll}
        1  & \mbox{if } n = 0 \\
        n \times (n-1)! & \mbox{otherwise}
      \end{array}
    \right.
\end{equation}
\end{document}
```

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

3

Jakld の 出力方法

NoStudent Left Behind

1. Command Prompt, cygwin で copy&paste
2. Shell の機能を使う
3. Output file を明示的に使用
   - (define out (open-output-file "outfile.txt"))
     out 同じファイル名のファイルがあると上書き
   - out
     #<port to outfile>
   - (display (fact 7) out) 結果を"outfile.txt"に書き込む
     5040
   - (newline out)
     #t
   - (close-output-port out)
4. call-with-output-file Output file の非明示的使用
   - (call-with-output-file "fact101.txt"
     (lambda (out)
     (newline out)
     (display (fact 101) out)
     (newline out)
     ))

4

10月22日・本日のメニュー

1.1 The Elements of Programming
1.1.1 Expressions
1.1.2 Naming and the Environment
1.1.3 Evaluating Combinations
1.1.4 Compound Procedures
1.1.5 The Substitution Model for Procedure Application
1.1.6 Conditional Expressions and Predicates
1.2 Procedures and the Processes They Generate
1.2.1 Linear Recursion and Iteration
1.2.2 Tree Recursion

NoStudent Left Behind

1.1.4 Compound Procedures (合成手続き)

NoStudent Left Behind

- "To square something, multiply it by itself."

➢ **(define (square x) (* x x))**

To square something, multiply it by itself.

- This is a compound procedure, of which name is "square". 手続き名前
- (define (<*name*> <*formal parameters*>)
  <*body*>) 仮パラメータ，本体
- (<*name*> <*parameters*>)
  procedure application 手続き適用

合成式(combinations)

- Primitives を使って式を合成
  - (+ 3 5)
    (演算子 引数 . . .)
- 評価法(実行方法)
  1. 部分式(subexpressions)を評価し，値を得る.
  2. 演算子(operator)に引数(arguments)を適用.
  - **procedure application (手続き適用)という**

Combination & Abstraction

- (* 5 5)
- (* 38 38)
- (* 389.2 389.2)

Abstraction of values

- (define foo 389.2)
- (* foo foo)
- (* foo (+ foo 5) 123)

Combination

- (+ 3 (* 53 12.4) 9)

Abstraction of combinations

```
(define (square x)
    (* x x))
```

- 正書法は下記

```
(define square
   (lambda (x) (* x x)) )
```

- 最初のは簡便記法: Syntax sugar (糖衣錠のこと)

式(expression)は評価されると値を返す

名前と値との連携

- `define` を使って式を合成
- `(define foo (+ 3 5))`
  `foo` の値は 8
- `(define bar +)`
  `bar` は + と同じ手続き
- `(bar 3 5)`

置換えモデルの例による説明

```
(define (square x) (* x x))
(define (sum-of-squares x y)
   (+ (square x) (square y)) )
(define (f a) (sum-of-squares (+ a 1) (* a 2)))
(f 5)                    「f の本体に5を適用」  仮パラメータ a を置換
     (sum-of-squares (+ a 1) (* a 2)) に a = 5 を適用
(sum-of-squares (+ 5 1) (* 5 2))   で a を 5 で置換(代入)
     (+ (square x) (square y))  に x = 6, y = 10 を適用
(+ (square 6) (square 10)) で x を 6 で, y を 10 で置換
     (* x x) に x = 6, (* x x) に x = 10 を適用
(+ (* 6 6) (* 10 10)) で x を 10 で置換
(+ 36 100)
136
```

1.1.6 Conditional Expressions (条件式)

- 絶対値

$$abs(x) = \begin{cases} x & \text{if } x > 0 \\ -x & \text{otherwise} \end{cases}$$

-
```
(define (abs x)
   (if (< x 0)
      (- x)
      x ))
```
- If : special form
- (if <*predicate*> <*consequent*> <*alternative*>)

述語　　帰結部　　代替部

1-2-1 Linear Recursion and Iteration

- 階乗の定義

```
(define (factorial n)
  (if (<= n 0)
      1
      (* n (factorial (- n 1))) ))
```

To define n!, if it is non-positive, return 1
otherwise, multiply it by (n-1)!

$$n! = n * (n-1)!$$

どう実行されるか。

Substitution model (置換モデル)で実行

factorial の置換モデルによる実行

NoStudent Left Behind

Linear recursive process （線形再帰的プロセス） Nに比例した数の再帰プロセスが生じる

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 (factorial 0)))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

14

factorial の計算量（Complexity）

NoStudent Left Behind

尺度が重要

factorialの呼ばれる回数
n回 for n! (time complexity)
未実行の * の量
最大 n回 for n! (space complexity)

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 (factorial 0)))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

deferred operationsを先に実行すれば，効率化が可能

末尾再帰（tail recursion）という

1-2-1 Linear Recursion and Iteration

末尾再帰（tail recursion）の効率実行

階乗の定義（その2）

To define n!, n! = 1 * 2 * ··· * n

*product = counter * product*
counter = counter + 1

```
(define (fact n)
  (fact-iter 1 1 n) )
(define (fact-iter product counter max-count)
  (if (> counter max-count)
      product
      (fact-iter (* counter product)
                 (+ counter 1)
                 max-count )))
```

どう実行されるか。Substation model (置換モデル) で実行

NoStudent Left Behind

factorial の置換モデルによる実行

```
(fact 6)
(fact-iter   1 1 6)
(fact-iter   1 2 6)
(fact-iter   2 3 6)
(fact-iter   6 4 6)
(fact-iter  24 5 6)
(fact-iter 120 6 6)
(fact-iter 720 7 6)
720
```

Linear iterative process
(線形反復プロセス)

tail recursion（末尾再帰）がiteration（反復）に !!!

fact から fact-iter へ自動変換が可能

fact-iter の呼ばれる回数
n回 for n! (time complexity)
未実行の演算の量はない
余分な仮パラメータ
2個 (space complexity)

以下のfactの構造 - Block Structure

```
(define (fact n)
   (define (iter product counter)
    (if (> counter n)
        product
        (iter (* counter product)
              (+ counter 1) )))
   (iter 1 1) )
```

- 手続きiterは，fact の中で有効。
- 仮パラメータ product,counterは，iterの中で有効
- 外部からは情報隠蔽(information hiding)
- これはオブジェクト指向の1つの特徴.

21

Tail recursion の補足説明

```
(define (f n)
  (if (<= n 0)
      1
      (* (f (- n 1)) n) ))
```

- このプログラムは次の翻訳
  n! = (n-1)! * n
- 下記のfactorialとの違いは

```
(define (factorial n)
    (if (<= n 0)
        1
        (* n (factorial (- n 1))) ))
```

22

factの置換モデルによる実行

NoStudent Left Behind

```
(f 6)
(* (f 5) 6)
(* (* (f 4) 5) 6)
(* (* (* (f 3) 4) 5) 6)
(* (* (* (* (f 2) 3) 4) 5
(* (* (* (* (* (f 1) 2) 3
(* (* (* (* (* (* (f 0) 1) 2) 3) 4) 5) 6)
(* (* (* (* (* (* 1 1) 2) 3) 4) 5) 6)
(* (* (* (* (* 1 2) 3) 4) 5) 6)
(* (* (* (* 2 3) 4) 5) 6)
(* (* (* 6 4) 5) 6)
(* (* 24 5))
(* 120)
720
```

末尾再帰ではない.

こういうプログラムは虫の一種!!!

処理系にやさしいプログラムを書くことが不可欠

Tail recursion による高速化

NoStudent Left Behind

```
SC> (time (null? (factorial 5000)))
total time: 0.729999999999563 secs
user time: 0.690993 secs
system time: 0 secs
#f
SC> (time (null? (f 5000)))
total time: 1.34000000000015 secs
user time: 1.321901 secs
system time: 0 secs
#f
SC> (time (null? (fact 5000)))
total time: 0.720000000001164 secs
user time: 0.701008 secs
system time: 0 secs
#f
```

コンパイルすると factorial （末尾再帰）は，fact（反復）は同じコードに変換される。インタプリタでも可.

2種類の階乗（n!）

NoStudent Left Behind

- トップダウン(top-down)式で計算-線形再帰

```
(define (factorial n)
  (if (<= n 0)
      1
      (* n (factorial (- n 1))) ))
```

- ボトムアップ(bottom-up)式で計算-線形反復

```
(define (fact-iter n)
  (define (iter product counter)
     (if (> counter n)
         product
         (iter (* counter product)
               (+ counter 1) )))
  (iter 1 1) )
```

スターリングの公式（Sterling's formula）

- n!は $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_n}$ ただし $\frac{1}{12n+1} < \lambda_n < \frac{1}{12n}$
- スターリング級数は

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{1}{51840n^3} - \frac{1}{2488320n^4} + \cdots\right)$$

- 対数をとると

$$\ln n! = n\ln n - n + \frac{1}{2}\ln(2\pi n) + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5} - \frac{1}{1680n^7} + \cdots$$

for n>>0

$$\ln n! \approx n\ln n - n$$

NoStudent Left Behind

THE POWER OF LOGARITHM

計算尺（slide rule, slipstick）
対数による積の計算
乗算→対数→加算
累乗→対数→乗算
2^{30} はいくら
2^{10} →対数→10log2 →3.01
$2^{10} \fallingdotseq 10^3$ →1K
$2^{30} \fallingdotseq 10^9$ →1G
音楽のピッチ
音の知覚

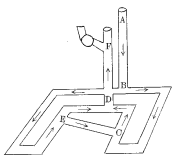
27

SLIDE RULE

NoStudent Left Behind

鉱山用通気計算尺
G.R.R.I.Mine Ventilation Slide Rule

一資源試型計算尺使用法説明書



ヘンミ計算尺株式会社
東京都千代田区神田駿河台４－４
３－２６３１（代）
（６９０６Ｎ）

- アポロ13
- ミクロの決死隊
- タイタニック

28

Scale Label	Value Relative to C/D
A	X^2
B	X^2
C	X
CF	X x pi
CF/M	X x $\log_{10}$
CI	1 / X
CIF	1 / (pi x X)
D	X
DF	X x pi
DF/M	X x $\log_{10}$
DFM	X x $\log_{10}$
DI	1 / X
DIF	1 / (pi x X)
E	e^x
H	-
H1, H2	Square root of $(1+X^2)$
HC	-

K	X^3
KZ	X x 360
L	log X
Lg	log X
Ln	ln X
LL0	$e^{0.001x}$
LL1 or ZZ1	$e^{0.01x}$
LL2 or ZZ2	$e^{0.1x}$
LL3 or ZZ3	e^x
LL00 or LL/0	e^{-x}
LL01 or LL/1	$e^{-0.1x}$
LL02 or LL/2	$e^{-0.01x}$
LL03 or LL/3	$e^{-0.001x}$
M	log X
P	Square root of $(1-X^2)$
p%	-
P1, P2	Square root of $(1-X^2)$

R	1 / X
R1, R2	Square root of X
S	sin X
Sh1, Sh2	sinh X
Sq1, Sq2	Square root of X
SRT	sin X, tan X
ST	sin, tan X
T	tan X, cotan X
T1, T3	tan X, cotan X
T2	tan X, cotan X
Th	tanh X
V	Volts
W1, W2	Square root of X
Z	X
ZZ1, ZZ2, ZZ3	e^x

大きな数・小さな数

NoStudent Left Behind

deca	da	$\times 10^1$
hecto	h	$\times 10^2$
kilo	K	$\times 10^3$
mega	M	$\times 10^6$
giga	G	$\times 10^9$
tera	T	$\times 10^{12}$
peta	P	$\times 10^{15}$
exa	E	$\times 10^{18}$
zetta	Z	$\times 10^{21}$
yotta	Y	$\times 10^{24}$

deci	d	$\times 10^{-1}$
centi	c	$\times 10^{-2}$
milli	m	$\times 10^{-3}$
micro	μ	$\times 10^{-6}$
nano	n	$\times 10^{-9}$
pico	p	$\times 10^{-12}$
femto	f	$\times 10^{-15}$
atto	a	$\times 10^{-18}$
zepto	z	$\times 10^{-21}$
yocto	y	$\times 10^{-24}$

30

10^1	ten *or* decad
10^2	hundred *or* hecatontad
10^3	thousand *or* chiliad
10^4	myriad
10^5	lac *or* lakh
10^6	million
10^7	crore
10^8	myriamyriad
10^9	milliard *or* billion
10^{12}	trillion
10^{15}	quadrillion
10^{18}	quintillion

10^{21}	sextillion
10^{24}	septillion
10^{33}	decillion
10^{63}	vigintillion
10^{303}	centillion
10^{100}	googol
10^{googol}	googolplex
10^N	Nplex
10^{-N}	Nminex

31

224plex 阿伽羅

88plex	無量大数	20plex	垓	4minex	絲 (糸)
80plex	不可思議	16plex	京	5minex	忽
72plex	那由他	12plex	兆	6minex	微
64plex	阿僧祇	8plex	億	7minex	纖 (繊)
56plex	恒河砂	4plex	萬(万)	8minex	沙
48plex	極	3plex	千	9minex	塵
44plex	載	2plex	百	10minex	埃
40plex	正	1plex	十	11minex	渺
36plex	澗	0plex	一	12minex	漠
32plex	溝	1minex	分	13minex	模糊
28plex	穣	2minex	厘	14minex	逡巡
24plex	杼 (禾偏)	3minex	毫(毛)	15minex	須臾

1minex	分	13minex	模糊
2minex	厘	14minex	逡巡 シュンジュン
3minex	毫 (毛) モウ	15minex	須臾シュユ
4minex	絲 (糸) シ	16minex	瞬息シュンソク
5minex	忽 コツ	17minex	弾指ダンシ
6minex	微 ビ	18minex	刹那
7minex	纖 (繊)セン	19minex	六徳 リットク
8minex	沙シャ	20minex	虚
9minex	塵ジン	21minex	空
10minex	埃アイ	22minex	清
11minex	渺ビョウ	23minex	浄
12minex	漠バク		

33

Procedure （手続き） vs. Process （プロセス）

- 手続きが再帰的とは、構文上から定義。
  自分の中で自分を直接・間接に呼び出す。
- 再帰的手続きの実行
  - 再帰プロセスで実行
  - 反復プロセスで実行
- 線形再帰プロセスは線形反復プロセスに変換可能
  「tail recursion（末尾再帰的）」
- 再帰プロセスでは、deferred operation用にプロセスを保持する必要あり ⇒ スペース量が余分に必要
- Scheme のループ構造はsyntactic sugar
  - **do, repeat, until, for, while**

NoStudent Left Behind

Applicative order vs. normal order (適用順序と正規順序)

- 「適用順序(作用順序, Applicative order)」
  今まで見てみた置換モデルの評価順序:
- 別の順序:「正規順序(normal-order)」:
  仮パラメータをすべて置換してから, 簡約する.

```
1.  (f 5)
2.       ((sum-of-squares (+ a 1) (* a 2)) 5)
3.  (sum-of-squares (+ 5 1) (* 5 2))
4.       ((+ (square x) (square y)) (+ 5 1) (* 5 2))
5.  (+ (square (+ 5 1)) (square (* 5 2))
6.       (+ ((* x x) (+ 5 1)) ((* x x) (* 5 2))
7.  (+ (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
8.  (+ (* 6 6) (* 10 10))
9.  (+ 36 100)
10. 136
```

2回同じものを計算

37

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 条件式の一般形; cond は特殊形式 (special form)
- (cond (⟨p1⟩ ⟨e11⟩ ...⟨e1m⟩)
  (⟨P2⟩ ⟨e21⟩...⟨e2k⟩)
  ...
  (⟨pn⟩ ⟨en1⟩...⟨enp⟩))
- 式の対 (⟨p⟩ ⟨e⟩ ... ⟨e⟩): 節(clause)
- ⟨p⟩: 述語(predicate)
- 述語の値: true (#t) か false (#f).
- ⟨e⟩: 帰結式(consequent expression)
- 特別の⟨p⟩: else (#t を返す)
- 節の評価は, ⟨p⟩が#tなら⟨e⟩が順に評価される.
- 一旦述語が#tを返すと, それ以降の節は評価されない.

38

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 絶対値をcase analysis (場合分け)で定義

$$|x| = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -x & \text{if } x < 0 \end{cases}$$

cond, if の実行は、
置換モデルで、特別扱い
Special form(特殊形式)

```
1.(define (abs x)
    (cond ((> x 0) x)
          ((= x 0) 0)
          ((< x 0) (- x)) ))
```

if : Syntax sugar

糖衣

```
2.(define (abs x)
    (cond ((< x 0) (- x))
          ((>= x 0) x) ))
```

```
3.(define (abs x)
    (cond ((< x 0) (- x))
          (else x) ))
```

```
(or (> x 0) (= x 0))
```

```
4.(define (abs x)
    (if (< x 0)
        (- x)
        x ))
```

39

1.1.6 Predicates (述語)

NoStudent Left Behind

- (and $\langle e_1 \rangle \ldots \langle e_n \rangle$) 論理積 (左から評価)
- (or $\langle e_1 \rangle \ldots \langle e_n \rangle$) 論理和 (左から評価)
- (not $\langle e \rangle$) 論理否定

例:

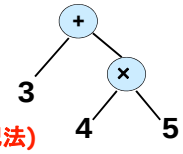
- $5 < x < 10 \Rightarrow$
- (define (>= x y)
  (or (> x y) (= x y)))
- (define (>= x y)
  (not (< x y)))

40

Ex.1.2 前置記法(prefix notation)

NoStudent Left Behind

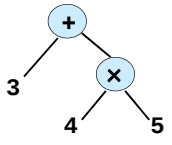
- 式 (演算子 被演算子 ...)
  operator operands
- 式の記法
  - 前置記法 (prefix notation , Pollish notation, ポーランド記法)
    + 3 * 4 5
  - 中置記法 (infix notation)
    3 + (4 * 5)
  - 後置記法 (postfix notation, reverse Pollish notation、逆ポーランド記法)
    3 4 5 * +
- 木表現はどれも同じ



41

木の辿り方から3つの記法への変換

- 木の辿り方
  - 前順走査 (pre-order traversal)
    ノード⇒左部分木⇒右部分木
    + ⇒ 3 ⇒ * ⇒ 4 ⇒ 5
  - 間順走査 (in-order tr.)
    左部分木⇒ノード⇒右部分木
    3 ⇒ + ⇒ 4 ⇒ * ⇒ 5
  - 後順走査 (post-order tr.)
    左部分木⇒右部分木⇒ノード
    3 ⇒ 4 ⇒ 5 ⇒ * ⇒ +



Javaプログラムのデモ

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/11/IntroAlgDs/

出席表への記入　（５分間）

NoStudent Left Behind

1. 反復型（繰返型）階乗のプログラムを書きなさい.

2. 本日の講義の感想を31文字でまとめなさい.

54

宿題：10月28日午後24時締切

NoStudent Left Behind

1. 反復型階乗プログラムのファイルを作成せよ.　fact-iter.scm
2. (fact-iter　100+ご自分の学籍番号の下1桁) を実行し，出力結果を求めよ.
3. 階乗のプログラムの説明と出力結果を latexで作成し，pdfを提出.
4. **教科書 1-2-3 ~ 1-3-1** を読み，①想定質問，②想定質問の解答，③その説明を記述．3の課題の後ろに書くこと.
5. Program ファイルとpdf (学籍番号-名前-回.pdf) を
   SICP-3@zeus.kuis.kyoto-u.ac.jp に送付

- **友達に教えてもらったら，その人の名前を明記すること．Webは出展を明記.**（otherwise『同じ』回答は減点）

55