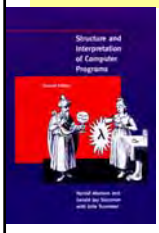
アルゴリズムとデータ構造入門
1.手続きによる抽象の構築
1.1 プログラムの構築

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の居室は文学部東館4階奥乃1研, 2研, 3研
平山 直樹(M1) 奥乃研・ロボット聴覚G
黄 楊暘(M1) 奥乃研・ロボット聴覚G
山口 雄紀(M1) 奥乃研・発達認知ロボティクスG

NoStudent Left Behind

教科書



Structure and Interpretation of Computer Programs (SICP)

- 世界中のComputer Scienceのトップレベルの教科書(過去20年間)
- 1回生後期で前半を
- 2回生前期で後半を(五十嵐先生)
- **MIT Press オンライン版**(無料)
- **Emacs Texinfo 形式**(無料)
- 日本語訳(邦訳・訳あり)約4.5K円

NoStudent Left Behind

教科書は持っているものとして進めます.

参考書とScheme 処理系

NoStudent Left Behind

1. **JAKLD Scheme(湯淺研開発・教育用計算機)**
   Java版(stand-alone, **携帯OK**), 他に Windows, Cygwin, Linux版あり.
   http://ryujin.kuis.kyoto-u.ac.jp/~yuasa/jakld/index-j.html
   Android版 (1回生の坂東君随意課題として作成)
   http://sigma-project.net/2011/01/25/androscheme/
2. **教育用計算機を使用. Install は不要.**
3. 他の処理系は**自己責任で使用**
4. ジョン・ベントリー(小林健一郎訳):『**珠玉のプログラミング ー本質を見抜いたアルゴリズムとデータ構造**』(ピアソン)
   英語版を.
5. **世界中にSICPのサイト・コースウェア等あり**
6. **宿題は自分でやること(先に答えを見ない)**
7. **Plagiarism(剽窃)は不正行為!**

成績評価(上限は壱萬點)

1. 試験　　70%
2. 必修課題　30%
   - ① 宿題で出した練習問題. Mail か レポート箱に提出
     (翌週同日12時締切, 工学部10号館1階)
   - ② 図形言語レポート(プログラムはメイルで提出)
3. 加算システム: 随意課題提出による "+α"
   - ① 第2章までのすべての練習問題
   - ② Fixed-Point探索過程のSchemeによる可視化
   - ③ アルゴリズムのSchemeによる可視化
   - ④ 抽象化によるSchemeプログラム(線形計画法, 整数論, 群論, 組合せ論, 古典力学, パズル解法, ゲーム, 数独)
   - ⑤ Lego Mindstorm用Lisp XS を使った自律ロボット
   - ⑥ 図形言語で circle-limit (難しいが提出者2名あり)
   - ⑦ 他の学生の支援

8

成績評価(上限は壱百點)

過去5年間の実績

本講義　受講者の

最終成績の平均点は

75点を超えています!

9

H24年度の目標 *No Student Left Behind*

落ちこぼれゼロ化作戦

TA3名が担当の学生の合格率向上を競う.

Modulo (学籍番号の下3桁, 3)

- = 0　平山君
- = 1　黄君
- = 2　山口君

11

講義内容理解の支援: *No Student Left Behind*

疑問等あれば,
教員・TAに質問を.
自分で考える癖を付けること.

NoStudent Left Behind

10号館4階TA居室
講義直前の昼休(12:45〜)
講義TAのWeb で
(http://winnie.kuis.kyoto-u.ac.jp/)

12

数理コース希望だから本科目は不要か

NoStudent Left Behind

1. 東京大学工学部計数工学科(数理情報学専攻)名誉教授
   杉原厚吉先生の主張
   『**数理工学にとってプログラミングは必須**』
2. 情報学での実験とは通常プログラミング.
3. **プログラミングができないとアイデア倒れ**.
4. **種々のプログラミング言語を使えること**は英語が大学院・企業で共通言語であるのと同様に, 文系・理系とも必須
5. 卒業特別研究の現状(奥乃の私見):
   - **数理工学コース**:**計算科学・シミュレーション工学を指向**.
     計算機科学コアの知識が不可欠.
   - **計算機科学コース**:**実はあまりプログラムを書かない**.
     最近はPython, Ruby, Javaが多そう.
   - **菊川怜さん(東大・工・建築):「遺伝的アルゴリズムを適用したコンクリートの要求性能型の調合設計に関する研究」**

13

10月2日・本日のメニュー

NoStudent Left Behind

1. SICP第1章　手続きによる抽象化 (abstraction)
2. 実例でSchemeプログラムを学ぶ
3. JAKLD (Scheme) の説明は10月9日 馬谷先生

OHPのマーク


教科書のまとめ


本講義独自

14

Lisp言語（Schemeの母体）

1. John McCarthyが1959年に設計・開発
   http://www-formal.stanford.edu/jmc/recursive.html
2. Fortran 言語についで2番目に古い言語
3. 種々の方言・実装あり、Schemeもその一つ
   MacLisp, Interlisp, TAO, Kyoto Common Lisp, …
4. 今日のオブジェクト指向などさまざまなアイデアを創出してきた「**原言語**」
5. 人工知能システム発達の**原動力**
6. 統合的プログラミング環境が提供
7. TRON（Disney）最初のCGによる映画
8. Pluto（134340）の軌道がChaoticの計算による証明
   - Galileo 以来の open problemの解決

16

John McCarthy, Prof. Emeritus, Stanford Univ

弟子さん：佐藤雅彦教授（情報），林晋教授（文学部，岩波文庫『ゲーデルの不完全性定理』訳者），居候：奥乃

Lisp によるプログラミング

1. 計算モデル：再帰方程式（recursion equation）という論理表現とその推論方式
2. Lispの方言－Scheme, CtCL, …
3. Lispの処理系(Implementation)
   - **解釈系（Interpreter）**: プログラムをそのまま解釈し、実行
   - **コンパイラ（Compiler）**：プログラムを機械語へ変換し、機械語をランタイムシステムの下で実行
4. 実装（implementation）
5. 手続き（procedure）であるプログラムとデータが同じ形

18

数学とコンピュータサイエンスの違い

NoStudent Left Behind

1. 宣言的知識(Declarative Knowledge)
   "What is true"という知識
   $\sqrt{x}$ *is the* y *such that* $y^2 = x$ *and* $y \geq 0$
2. 規範的知識(命令的, Imperative Knowledge)
   "How to"という知識

$x = 2$ に対する $\sqrt{x}$ の値を求めるには

予測 (guess)	相棒は商で求まる	予測とその相棒の平均値で改善
1	2/1 = 2	(1 + 2)/2 = 1.5
1.5	2/1.5 = 1.333	(1.5 + 1.333)/2 = 1.4167
1.4167	2/1.4167 = 1.4118	(1.4167 + 1.4118) = 1.4142
1.4142	2/1.4142 =	

19

"How to" 知識を概念化する

NoStudent Left Behind

1. 手続き(procedure)
   所望の値を求めるステップ系列の概念ーrecipe
2. 計算プロセス(computational process)
   具体的に計算機の中で実行されるステップの展開
   ー実際の調理
3. データ(data) ー 材料
4. プログラム(program)= 手続き ＋ データ
   計算プロセスはプログラム指示によりデータを操作
5. 言語(language) あるいはプログラミング言語
   計算プロセスを記述するために使用
6. 指示誤り:バグ(虫, bug)、スリップ(glitches)
7. 間違い修正:虫とり(debug)

20

"How to" 知識・概念化のポイント

NoStudent Left Behind

言語(language)

複雑さとの戦いー単純なデータと手続き

- Vocabulary (語彙)
- Syntax (構文)ー 複合式を構築するためのルール
- Semantics (意味)ー構成子に意味を付与するためのルール

1. 手続き抽象化(procedure abstraction)
2. データ抽象化(data abstraction)

21

Intermission

Walt Disneyの映画

“TRON”

をご存じですか？

NoStudent Left Behind

TRON Disney 映画 (1982)

A masterpiece of breakthrough CGI ingenuity, Disney celebrates the 20th anniversary of TRON, a dazzling film at the flashpoint of a continuing revolution in its genre. This special collector's edition showcases an epic adventure inside a brave new world where the action is measured in microseconds.

When Clu (Kelvin Flynn; Jeff Bridges) hacks the mainframe of his ex-employer to prove his work was stolen by another executive, he finds himself on a much bigger adventure. Beamed inside by a power-hungry master control program, he joins computer gladiators on a deadly game grid, complete with high-velocity "light cycles" and Tron (Alan Bradley; Bruce Boxleitner), a specialized security program. Together, they fight the ultimate battle with the MCP to decide the fate of both the electronic world and the real world!


NoStudent Left Behind

TRON のベース ASAS on Lisp

Craig W. Reynolds (III): Computer animation with scripts and actors, *Computer Graphics*, Vo.16, No.3, pp.289-296.

```
(defop arch-fractalizer
   (param: arch-element top-color bot-color levels
           fractal-ratio height width leg-width)
   (local: (total-levels  levels)
           (offset-dist  (half (dif width leg-width)))
           (sub-tower-offset-1 (vector offset-dist 0 0))
           (sub-tower-offset-2 (mirror x-axis
                                sub-tower-offset-1)))
   (arch-tower levels))

(defop arch-tower
   (param: levels)
   (if (zerop levels)
       (then nothing)
       (else (add-arch-level (arch-tower
                              (dif levels 1))))))
(defop add-arch-level
    (param: sub-tower)
    (grasp sub-tower
           (scale factal-ratio)
           (move (vector 0 height 0))
           (rotate 0.25 y-axis))
    (grasp arch-element
           (recolor (interp (quo levels total-levels)
                            bot-color top-color)))
    (subworld (group arch-element
              (move subtower-offset-1 sub-tower)
              (move subtower-offset-2 sub-tower)))))
```


NoStudent Left Behind

Intermission

NoStudent Left Behind

Pluto（冥王星）が惑星(planet) から準惑星(dwarf planet) に変更になったのをご存じですか？

27

Pluto(134340)

NoStudent Left Behind

"Chaotic Evolution of the Solar System", Gerald Jay Sussman and Jack Sisdom, *Science*, 257, July 1992.

The evolution of the entire planetary system has been numerically integrated for a time span of nearly 100 million years. This calculation confirms that the evolution of the solar system as a whole is chaotic, with a time scale of exponential divergence of about 4 million years. Additional numerical experiments indicate that the Jovian planet subsystem is chaotic, although some small variations in the model can yield quasiperiodic motion. The motion of Pluto is independently and robustly chaotic.

証明は場合分けで行う．Lispで．

Intermission

NoStudent Left Behind

東京大学工学部機械情報学科の稲葉研究室では，ロボットの制御はすべてLISPで書かれています．

因みに，稲葉研は前任の井上名誉教授時代から世界のHumanoidsロボット研究の先駆者です．

世界的には，ROS (Robot Operating System) + Python/C++ が主流になりつつあります．

29

10月2日・本日のメニュー

1. オリエンテーション
2. 出席確認
3. SICP第1章　手続きによる抽象化 (abstraction)
4. 実例でSchemeプログラムを学ぶ

【注意】 軽いノリですぐにやってみる

できない理由をあれこれ考えない

階乗のプログラムを書いてみよう

- 数学の記法で書くと

  n!=

階乗のプログラムを書いてみよう

1. fact(n) = 1 * 2 * 3 * ··· * n
2. fact(n) = n *(n-1) *(n-2) * ···*1
3. fact(n) = 1 if n ≦ 0
   n*fact(n-1) otherwise
4.
```
(define (fact n)
  (if (<= n 0)
      1
      (* n (fact (- n 1))) ))
```
5. (fact 3)
6. (fact 10)
7. (fact 100)

1.1.4 Compound Procedures (合成手続き)

- "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"**.
- **(define (<*name*> <*formal parameters*>)**
  <*body*>)
  - <*name*> 名前
  - <*formal parameter*> 仮パラメータ
  - <*body*> 本体

NoStudent Left Behind

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

NoStudent Left Behind

階乗のプログラムを書いてみよう

NoStudent Left Behind

$$factorial(n) = \begin{cases} 1 & \text{if } n \leq 0 \\ n * factorial(n-1) & \text{otherwise} \end{cases}$$

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

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

1. `(factorial 3)`
2. `(factorial 10)`
3. `(factorial 100)`
```

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

NoStudent Left Behind

置換モデルの例による説明

NoStudent Left Behind

```
(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)) )

1.(f 5)                          「f の本体に5を適用」
2.   (sum-of-squares (+ a 1) (* a 2))「f の本体」
3.    仮パラメータ a を 置換
4.(sum-of-squares (+ 5 1) (* 5 2))
              「sum-of-squares の本体に 6 と10 を適用」
5.(+ (square 6) (square 10))
6.(+ (* 6 6) (* 10 10))
7.(+ 36 100)
8.136
```

44

1-2-1 Linear Recursion and Iteration

- 階乗の定義（その1）

```
(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)!

- どう実行されるか。Substation model で実行
- **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
- 積は **deferred operations（遅延演算）**

NoStudent Left Behind

factorial の置換モデルによる実行

```
(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
```


factorial の置換モデルによる実行

```
(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 operation（未実行演算）

保持用の空間が必要（空間計算量）

factorial の計算量（Complexity）

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
```

LaTeX でレポートを書く

NoStudent Left Behind

¥documentclass{a4paper,12pt}{article}
¥usepackage{listings}
¥begin{document}
¥lstset{numbers=left,basicstyle=¥small}
¥lstinputlisting{fact.scm}

以上で，ファイル "fact.scm" に書かれたプログラムのリスティングが得られる．その下にプログラムの説明を書く．

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/12/IntroAlgDs/listing.tar.gz にサンプルあり．

¥end{document}

50

宿題：10月9日正午締切

NoStudent Left Behind

1. 階乗のプログラムのファイルを作成せよ．factorial.scm
2. 階乗のプログラムを実行し，出力結果を求めよ．
   1. (load "factorial.scm")
   2. (factorial "100+ご自分の学籍番号の下1桁")
3. 階乗のプログラムの説明と出力結果を latexで作成し，pdfをレポートとする．（紙でも可）
4. Program ファイルとpdf を SICP-1@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）

51