アルゴリズムとデータ構造入門

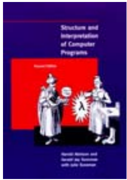
1.手続きによる抽象の構築

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/06/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

TAの居室は10号館4階奥乃1研，2研
糸山 克寿 (mod(学籍番号, 3) ≡0) 奥乃研・音楽G
武田 龍 (mod(学籍番号, 3) ≡1) 奥乃研・ロボット聴覚G
福林 雄一郎 (mod(学籍番号, 3) ≡2) 奥乃研・音声対話G

教科書 Structure and Interpretation of Computer Programs (SICP)



1. 世界中のComputer Scienceのトップレベルの教科書
2. 1回生後期で前半を
3. 2回生前期で後半を（湯淺先生）
4. MIT Press 提供オンライン版（無料）
5. Emacs Texinfo 形式（無料）
6. 日本語訳（邦訳・訳あり）約4.5K円

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

1. ジョン・ベントリー（小林健一郎訳）:『珠玉のプログラミング—本質を見抜いたアルゴリズムとデータ構造』（ピアソン）
2. 世界中にSICPのサイト・コースウェア等あり
3. 宿題は自分でやること（答えを見ない）
4. TUT Scheme（湯淺研開発・メディアセンタ）
   Windows, Cygwin, Linux最新版×，あり
5. 他の処理系は自己責任で使用のこと
   1. MIT-Scheme-6001
   2. Dr.Scheme, Chez Scheme, ・・・

成績評価

1. **試験 80%**
2. **必修課題 20%**
   ① 宿題で出した練習問題. レポート箱に提出（翌週前日17時締切, 工学部10号館1階, 今出川裏通り東端南）
   ② 図形言語レポート（プログラムはメイルで提出）
3. **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 これはすごいという抽象化を使った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（難しいが提出者あり）
   ⑦ 他の学生の支援

Computer Science を極めるには

1. 『計算機プログラムの構造と解釈』（SICP）を読む
2. 『コンピュータの構成と設計ーハードウェアとソフトウェアのインタフェース』（上下）を読む.
3. 『珠玉のプログラミング』を読む
4. 情報処理技術者試験を受ける
   ① 2回生迄に**基礎情報処理技術者試験**に合格
   ② 3回生迄に**ソフトウェア開発技術者試験**に合格
5. 自分の腕を磨く
   ① ACM国際大学プログラミングコンテストに出場
   ② Lego Mindstorm, ・・・ で遊ぶ

10月3日・本日のメニュー

1. **いきなりScheme**
2. SICP第1章 手続きによる抽象化(abstraction)
3. SICP 1.1 プログラムの要素
4. TUT Scheme の説明は10月10日 平石君（湯淺研D2)

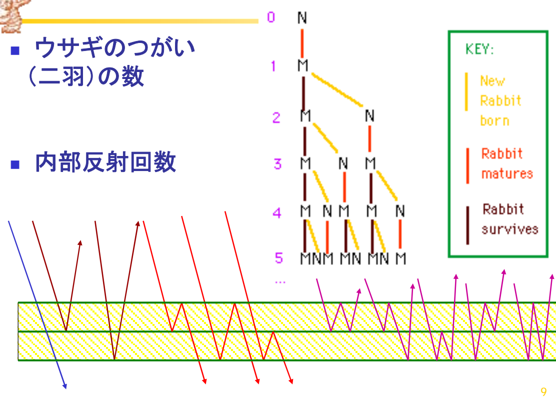
階乗のプログラムを書いてみよう

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)

8

フィボナッチ関数(Fibonacci Function)

- ウサギのつがい(二羽)の数
- 内部反射回数



9

フィボナッチ数のプログラムを書こう

1. fib(n) = 1 if n ≦ 1
   fib(n-1)+fib(n-2) otherwise
2.
```
(define (fib n)
   (if (<= n 1)
       1
       (+ (fib (- n 1))
          (fib (- n 2)) )))
```
3. (fact 3)
4. (fact 10)
5. (fact 30)

10

アッカーマン関数を知っていますか

1. ack(m, n) =
   n+1 if m = 0
   ack(m-1, 1) if n = 0
   ack(m-1, ack(m, n-1)) otherwise
2.
```
(define (ackermann m n)
  (if (= m 0)
      (+ n 1)
      (if (= n 0)
          (ackermann (- m 1) 1)
          (ackermann (- m 1)
                (ackermann m (- n 1))
          ))))
```
3. (ackermann 0 2)
4. (ackermann 1 2)
5. (ackermann 2 2)
6. (ackermann 3 2)

宿題：アッカーマン関数の値

提出先は10号館レポート箱,
締切10日10時30分

- (ackermann 0 2)
- (ackermann 1 2)
- (ackermann 2 2)
- (ackermann 3 2)

計算過程を書くこと. 以下随意課題

1. (ackermann 0 n) ≡ n+1
2. (ackermann 1 n) ≡ ?
3. (ackermann 2 n) ≡ ?
4. (ackermann 3 n) ≡ ?

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の方言ー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（implement)
5. 手続き（procedure)であるプログラムとデータが同じ形

Lisp言語

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. 統合的プログラミング環境が提供
6. TRON(Disney)最初のCGによる映画
7. Pluto(134349)の軌道がChaoticの計算による証明
   - Galileo 以来の open problemの解決

“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 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 (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!

“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)))))
```

Figure 1: An Arch Fractal

Copyright © 1982 by Walt Disney Productions and Information International, Inc.

Figure 3: Solar Sailer Escape Sequence from TRON

Pluto（134349）

"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.

20

数学とコンピュータサイエンスの違い

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*

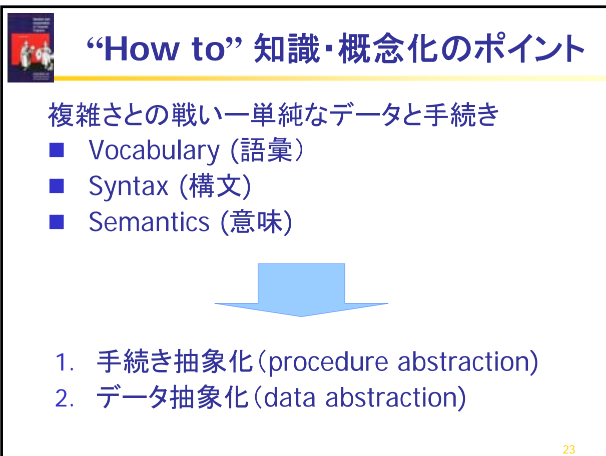
予測 (guess)	相棒は商で求まる	予測とその相棒の平均値で改善
1	2/1 = 2	(1 + 2)/2 = 1.5
1.5	2/1.5 = 1.333	(1.5 + 1.333)/2 = 1.4167
1.4167	2/1.4167 = 1.4118	(1.4167 + 1.4118) = 1.4142
1.4142	2/1.4142 =	

21

“How to” 知識を概念化する

1. **手続き（procedure）**
   所望の値を求めるステップ系列の概念 －recipeのようなもの
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－実際の調理
3. **データ（data）** － 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. 指示誤り:バグ（虫，bug）、スリップ（glitches）
6. 間違い修正：虫とり（debug）
7. 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
   - **Vocabulary (語彙）**
   - **Syntax (構文)**－ 複合式を構築するためのルール
   - **Semantics (意味)**－構成子に意味を付与するためのルール

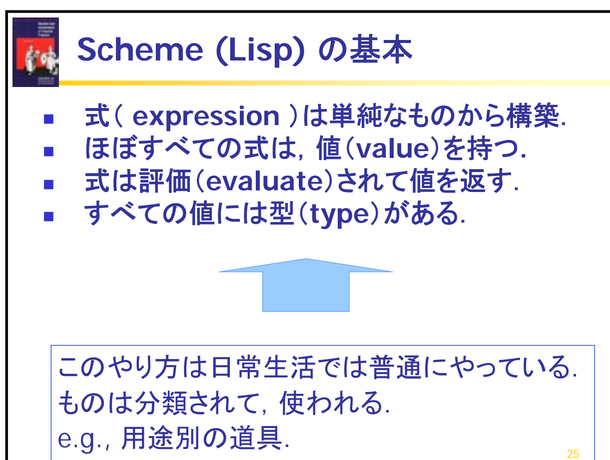
22

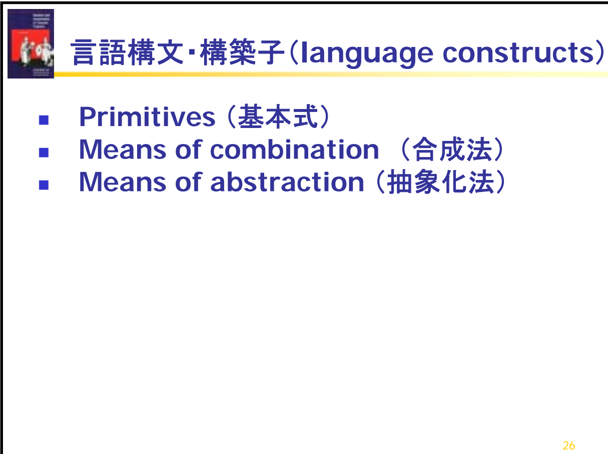


1.1 言語の要素

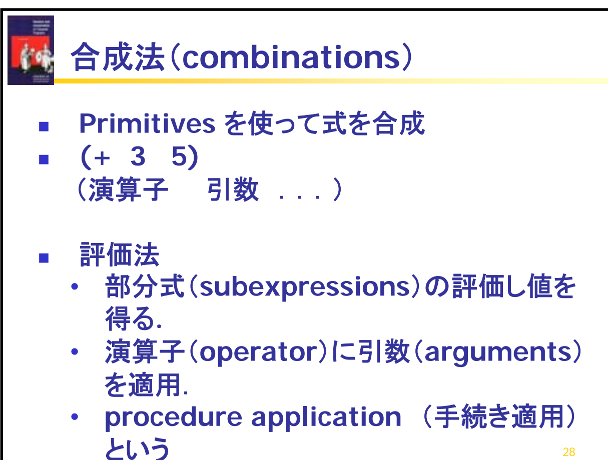
- **expressions (式)**
  primitives(基本式)とcombinations(合成式)で構成
- **means of abstraction(抽象化法)**
- **Creating procedure objects (手続きの作成法)**
- **Viewing the rules of evaluation from a computational perspective (計算という観点からの評価法)**

24




言語構文・構築子(language constructs)
Primitives (基本式)
Means of combination (合成法)
Means of abstraction (抽象化法)
26

基本式(primitives)
Self-evaluating primitives(評価すると自分自身の値)
Numbers (数): 38, 3.80, 1.4141, 2.3e-4, 3/5
Strings (文字列): “moji”, “a”,
Booleans (論理式): #t, #f
Built-in procedure (組込み手続き)
基本オブジェクト(primitive objects)の処理
Numbers(数): +, -, *, /, <, =, <=, ...
Strings(文字列): string-length, string=?
Boolenas(論理式): and, or, not, xor, nand,
Names for built-in procedures
+ ⇒ +という組込み手続き
27


合成法(combinations)
Primitives を使って式を合成
(+ 3 5)
(演算子 引数 . . .)
評価法
部分式(subexpressions)の評価し値を得る.
演算子(operator)に引数(arguments)を適用.
procedure application (手続き適用)という
28

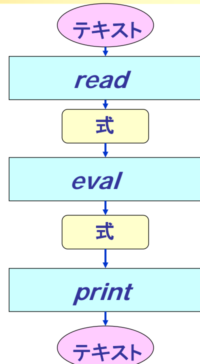
名前と値との連携

- 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)

29

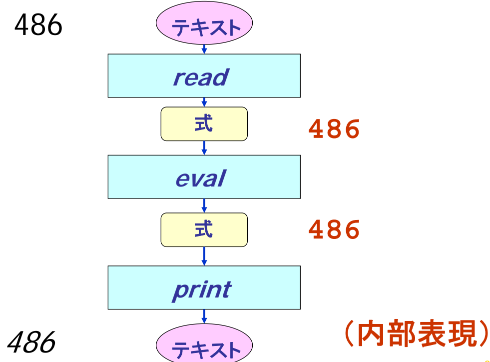
1.1.1 Expressions(式)

- An interpreter (解釈系):
  read-evaluate-print ループ (REPL) を繰り返す
- read: テキスト表現の式を内部表現に変換
- evaluate:式を評価
- print: 評価結果の式を内部表現からテキスト表現に変換, 出力

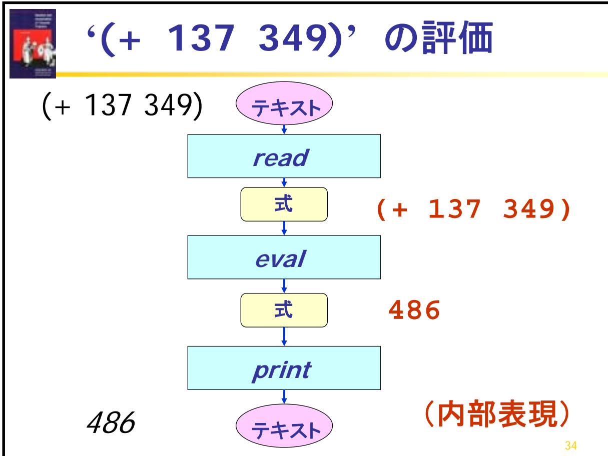


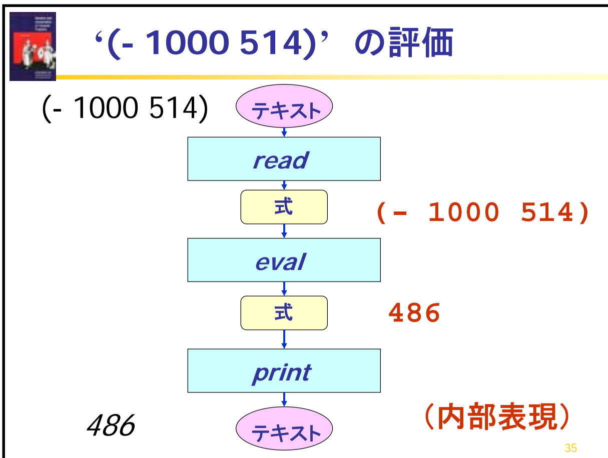
32

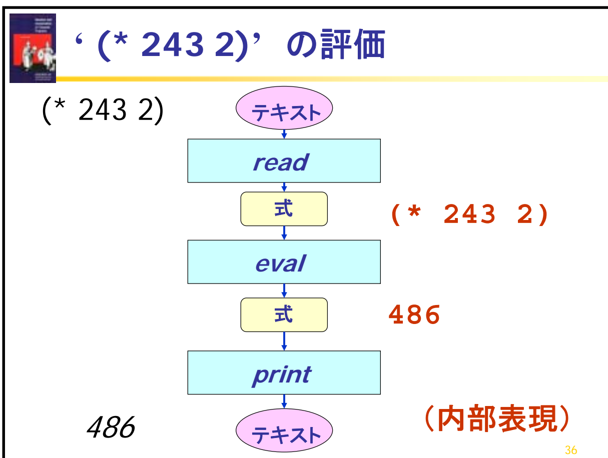
'486' の評価

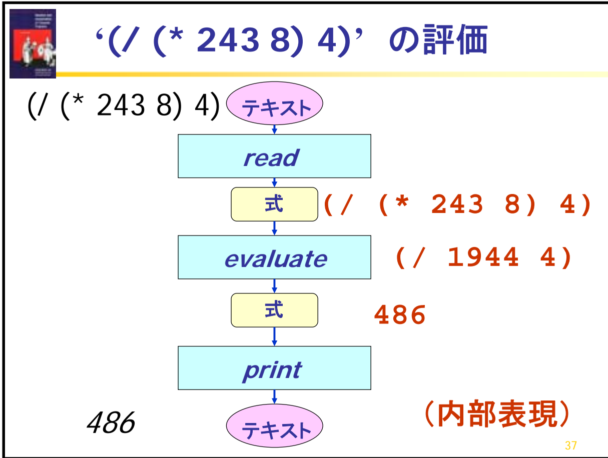


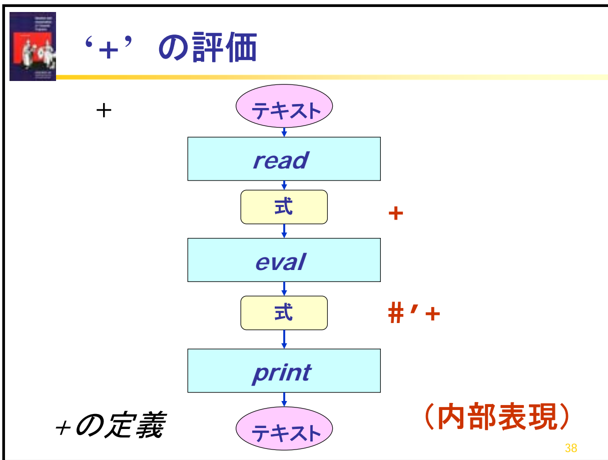
33


‘(+ 137 349)’ の評価
(+ 137 349)
テキスト
read
式
(+ 137 349)
eval
式
486
print
486
テキスト
（内部表現）
34


‘(- 1000 514)’ の評価
(- 1000 514)
テキスト
read
式
(- 1000 514)
eval
式
486
print
486
テキスト
（内部表現）
35


‘ (* 243 2)’ の評価
(* 243 2)
テキスト
read
式
(* 243 2)
eval
式
486
print
486
テキスト
（内部表現）
36





1.1.1 Expressions（式）

- 486
- (+ 137 349)
- (- 1000 334)
- (* 5 99)
- (/ 10 5)
- (+ 2.7 10)

39

1.1.1 Expressions(式)

```
(+ 21 35 12 7)

(* 25 4 12)

(+ (* 3 5) (- 10 6))

(+ (* 3 (+ (* 2 4) (+ 3 5)))
   (+ (- 10 7) 6) )

(+ (* 3
      (+ (* 2 4) (+ 3 5)) )
   (+ (- 10 7)  6))
```

40

1.1.2 Naming (名前) Environment (環境)

- The critical aspect of a programming language is the means it provides for using a name to refer to computational object.
- The name identifies a **variable** whose **value** is the object.
- **What is a computational object?**
  - from simple data such as numbers
  - to Complex structures
- The **environment** provides some sort of memory that keeps track of the name-object pairs.

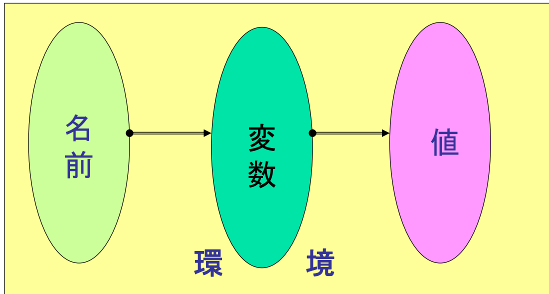
41

1.1.2 Naming and the Environment

- (define size 2)　処理系依存
- size
  2
- (* 5 size)
  10
- (define pi 3.14159)
- (define radius 10)
- (* pi (* radius radius))
  314.159
- (define cirumference (* 2 pi radius))
- circumference
  62.8318

42

1.1.2 Naming and the Environment

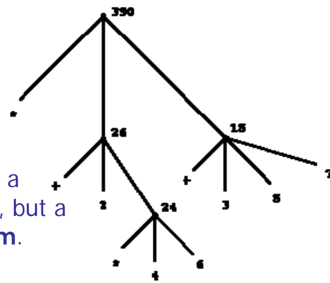


Global environment (大域環境)

43

1.1.3 Evaluating Combinations

- (* (+ 2 (* 4 6))
  (+ 3 5 7))



- define is not a combination, but a **special form**.

45

1.1.4 Compound Procedures(合成手続き)

- The rule of *procedure definitions*

1. Numbers and arithmetic operations are primitive data and procedures.
2. Nesting of combinations provides a means of combining operations.
3. Definitions that associate names with values provide a limited means of abstraction.

- Use define to associate a procedure with a name:
- “To square something, multiply it by itself.”

46

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*>)
  - <formal parameter> 仮パラメータ
  - <body> 本体

47

1.1.4 Compound Procedures(合成手続き)

- (square 21)
  441
- (square (+ 2 5))
  49
- (square (square 3))
  81
- (define (sum-of-square x y)
  (+ (square x) (square y)))
- (sum-of-square 3 4)
  25
- (define (foo a)
  (sum-of-squares
  (+ a 1)
  (* a 2)))
- (foo 5)
  136

Compound procedures are used in exactly the same way as primitive procedures. The latter is built into the interpreter.

48

1.1.5 The Substitution Model for Procedure Application(手続き適用の置換モデル)

- Assume that the mechanism for applying primitive procedures to arguments is built into the interpreter.
- For compound procedures, the application process is as follows:

To apply a compound procedure to arguments, evaluate the body of the procedure with each formal parameter replaced by the corresponding argument.

49

1.1.5 The Substitution Model for Procedure Application(手続き適用の置換モデル)

- (foo 5)

1. foo is the procedure defined before.
2. By retrieving the body of foo
   (sum-of-squares (+ a 1) (* a 2))
3. Replace the formal parameter a by the argument 5: (sum-of-squares (+ 5 1) (* 5 2))
4. Two arguments are evaluated and substituted for the formal parameters x and y:
   (+ (square 6) (square 10))
5. (+ 36 100)
6. 136

50

1.1.5 The Substitution Model for Procedure Application(手続き適用の置換モデル)

- A model that detemines the meaning of procedure application, insofar as the procedure in this chapter are concerned.
- The purpose of the substitution is just for explanation.
- The substitution model is the first step and we will present a sequence of increasingly elaborate models of how interpreters work.

51

1.1.5 The Substitution Model 注意

- (|-∀) ∀右 と (∃|-) ∃左 は、eigenvariable condition が必要。要注意。
- a = b |- a = b　　　　　a = b |- a = b
  ------------------ (|-∀)　----------------------(∃|-)
  a = b × ∀x.(x = b)　　　∃x.(x = b) × a = b

  上記は成立しない！

- a = b |- a = b　　　　　a = b |- a = b
  -------------------- (|-∃)　--------------------(∀|-)
  a = b |- ∃x.(x = b)　　　∀x.(x = b) |- a = b

  は成立！

52

Applicative order vs. normal order (作用的順序と正規順序)

- 1.1.3: ***applicative-order evaluation***
  "*Evaluate the arguments and then apply*" method called
- **Alternative evaluation model:** ***normal-order evaluation***
  "*fully expand and then reduce*"
- Exercise 1.5 is an instance of an illegitimate value where these two evaluation methods do not give the same result.

53

Alternative Model for Procedure Application

- (foo 5)

1. (sum-of-squares (+ 5 1) (* 5 2))
2. (+ (square (+ 5 1)) (square (* 5 2)))
3. (+ (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
4. (+ (* 6 6) (* 10 10))
5. (+ 36 100)
6. 136

The same answer, but different process.
The evaluations of (+ 5 1) and (* 5 2) are each performed twice.

54

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

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

- (define (abs x)

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

- General form of a conditional expression

55

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

- General form of a conditional expression
- ```
  (cond (<p1> <e1>)
        (<P2> <e2>)
        …
        (<pn> <en>) )
  ```
- A pair of expressions (<p> <e>) called clauses.
- <p> predicate. Its value is interpreted as either **true** or **false**.
- <e> consequent expression
- Special <p>: **else**

56

1.1.6 Condtional Expressions (条件式)

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

57

1.1.6 Predicates (述語)

- (**and** <e_1> … <e_n>) 論理積(左から評価)
- (**or** <e_1> … <e_n>) 論理和(左から評価)
- (**not** <e>) 論理否定

例:

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

58
  ```

宿題1:アッカーマン関数の値

提出先は10号館レポート箱,
締切10日10時30分

- (ackermann 0 2)
- (ackermann 1 2)
- (ackermann 2 2)
- (ackermann 3 2)

計算過程を書くこと. 以下随意課題

1. (ackermann 0 n) ≡ n+1
2. (ackermann 1 n) ≡ ?
3. (ackermann 2 n) ≡ ?
4. (ackermann 3 n) ≡ ?

59

宿題: 10月10日10時30分締切

- [必修] 問題1.1～1.4
- [随意] 問題1. 5は難しい.
- レポート用紙に読める字で書くこと.
- ワープロ可. その場合名前等もワープロで.
- 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』回答は減点します)

61