

アルゴリズムとデータ構造入門

2.4 抽象データの複数の表現法 (Multiple Representations for Abstract Data)

奥乃 博



The Eighth Commandment

Use help function to abstract from representations.

The Ninth Commandment

Abstract common patterns with a new function.

The Tenth Commandment

Build functions to collect more than one value at a time.

(Friedman, et al. "The Little Schemer", MIT Press)



12月13日・本日のメニュー

- 2.3.3 Representing Sets (集合)
 - Sets as unordered lists & as ordered lists (補足)
 - Sets as binary trees
 - Sets and information retrieval
- 2.4 Multiple Representations for Abstract Data
 - 2.4.2 Representations for Complex Numbers
 - 2.4.2 Tagged data
 - 2.4.3 Data-Directed Programming and Additivity
 - 配布する用紙に名前を記入して下さい。
 - 今週の宿題と共に提出して下さい。

順序有無による集合演算コスト差

1. 順序なし表現

U (8 2 10 0 6 4) 8 10 0 6

(5 3 2 1 4)

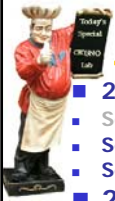
$\Theta(n^2)$ $n = \max\{m1, m2\}$ ($m1 * m2$ のオーダー)

2. 順序付き表現

U (0 2 4 6 8 10) 0 1 2 3

(1 2 3 4 5)

$\Theta(n)$ $n = \max\{m1, m2\}$ ($m1 + m2$ のオーダー)



12月13日・本日のメニュー

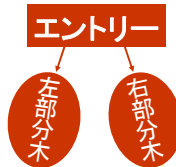
- 2.3.3 Representing Sets (集合)
 - Sets as unordered lists & as ordered lists (補足)
 - Sets as binary trees
 - Sets and information retrieval
- 2.4 Multiple Representations for Abstract Data
 - 2.4.2 Representations for Complex Numbers
 - 2.4.2 Tagged data
 - 2.4.3 Data-Directed Programming and Additivity
- 配布する用紙に名前を記入して下さい。
- 今週の宿題と共に提出して下さい。

16



集合の二進木(binary tree)表現

- リスト構造(木)で集合を表現
- 設計方針
 - ・ 木の高さをhとすると、 $\Theta(h^2)$ の計算量がかかるので、何らかの順序関係を導入する必要がある。
 - ・ 左部分木のエンタリーはノードのそれより大きくない
 - ・ 右部分木のエンタリーはノードのそれより大きい
- ノードの表現法
 - ・ 次のリストでノードを表現 (エンタリー 左部分木 右部分木)

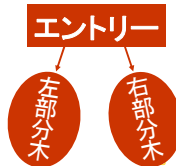




二進木(binary tree)表現の実装

```
(define (entry tree) (car tree))
(define (left-branch tree) (cadr tree))
(define (right-branch tree) (caddr tree))

(define (make-tree entry left right)
  (list entry left right) )
```



二進木表現の曖昧性

集合{1, 2, 3, 4, 5, 6} の二進木表現

19

集合(set)のbinary tree表現

```

(define (element-of-set? x set)
  (cond ((null? set) false)
        ((= x (entry set)) true)
        ((< x (entry set))
         (element-of-set? x (left-branch set)))
        (else
         (element-of-set? x (right-branch set)))))

(define (adjoin-set x set)
  (cond ((null? set) (make-tree x () ()))
        ((= x (entry set)) set)
        ((< x (entry set))
         (make-tree (entry set)
                    (adjoin-set x (left-branch set))
                    (right-branch set)))
        (else
         (make-tree (entry set)
                    (left-branch set)
                    (adjoin-set x (right-branch set)))))
  )

```

20

adjoin-set の動き

3, 2, 1, 5, 4, 6	3, 4, 2, 5, 6, 1	4, 2, 1, 5, 6, 3
------------------	------------------	------------------

21

adjoin-set の動き

6,5,4,3,2,1 6 1 1,2,3,4,5,6

22

balanced binary tree表現

```

(define (tree->list-1 tree)
  (if (null? tree)
      ()
      (append
       (tree->list-1 (left-branch tree))
       (cons (entry tree)
             (tree->list-1 (right-branch tree))))))

(define (tree->list-2 tree)
  (define (copy-to-list tree result-list)
    (if (null? tree)
        result-list
        (copy-to-list
         (left-branch tree)
         (cons (entry tree)
               (copy-to-list (right-branch tree)
                             result-list )))))
  (copy-to-list tree ()))

```

23

balanced binary tree表現

```

(define (list->tree elements)
  (car (partial-tree elements (length elements))))

(define (partial-tree elts n)
  (if (= n 0)
      (cons () elts)
      (let ((left-size (quotient (- n 1) 2)))
        (let ((left-result (partial-tree elts left-size)))
          (let ((left-tree (car left-result))
                (non-left-elts (cdr left-result))
                (right-size (- n (+ left-size 1))))
            (let ((this-entry (car non-left-elts))
                  (right-result (partial-tree
                                (cdr non-left-elts)
                                right-size)))
              (let ((right-tree (car right-result))
                    (remaining-elts (cdr right-result)))
                (cons (make-tree this-entry
                                left-tree
                                right-tree
                                remaining-elts))))))))))

```

24



balanced binary tree表現(改)

```

(define (list->tree elements)
  (car (partial-tree elements (length elements))))
(define (partial-tree elts n)
  (if (= n 0)
      (cons () elts)
      (let* ((left-size (quotient (- n 1) 2))
             (left-result (partial-tree elts left-size))
             (left-tree (car left-result))
             (non-left-elts (cdr left-result))
             (right-size (- n (+ left-size 1)))
             (this-entry (car non-left-elts))
             (right-result
              (partial-tree (cdr non-left-elts)
                            right-size )))
            (right-tree (car right-result))
            (remaining-elts (cdr right-result)))
        (cons
         (make-tree this-entry left-tree right-tree)
         remaining-elts ))))

```

25



Sets and information retrieval

```

(define (lookup given-key set-of-records)
  (cond ((null? set-of-records) false)
        ((equal? given-key (key (car set-of-records)))
         (car set-of-records))
        (else (lookup given-key (cdr set-of-records))))
  ))

```

26



簡単な情報検索

```

(define (lookup given-key set-of-records)
  (cond ((null? set-of-records) false)
        ((equal? given-key (key (car set-of-records)))
         (car set-of-records))
        (else (lookup given-key (cdr set-of-records)))))

(define population
  '((China 1285.0 660.5 624.5)
    (India 1025.1 528.5 496.6)
    (USA 285.9 141.0 144.9)
    (Indonesia 214.8 107.8 107.1)
    (Brazil 172.6 85.2 87.4)
    (Pakistan 145.0 74.5 70.5)
    (Russia 144.7 67.7 77.0)
    (Bangladesh 140.4 72.3 68.0)
    (Japan 127.1 62.2 65.0)
    (Nigeria 116.9 59.0 58.0)
    (Mexico 100.4 49.6 50.7) ))

(lookup 'Japan population)

```

27



key の順序

1. 数
 1. 昇順(increasing order, ascending order) <
 2. 降順(decreasing order, descending order) >
2. 辞書式順序(lexicographical order)
 1. (string=? "PIE" "pie")
 2. (string-ci=? "PIE" "pie")
 3. string<?, string<=??, ...
 4. char=? , char-ci=? , char>? , char>=? , ...
3. alphanumeric order

28



ソーティングの応用

1. 本1冊に出てくる単語の頻度を求めよ。
2. Unix の pipe で処理
次の1行のコマンドでできる。

```
tr '[ ¥t,.;:]*' '¥n' < file |
tr '[A-Z]' '[a-z]' | sort |
uniq -c | sort -r
```
3. www.gutenberg.org よりフルテキストを入手、
Gulliver's Travel
the 2894, of 1844, and 1755, to 1557,
i 1311, a 1177, in 984, my 768, was 625
TAO
the 675, and 373, to 345, of 335, is 290
it 225, not 164, in 154, he 136, a 136

29



12月13日・本日のメニュー

- 2.3.3 Representing Sets (集合)
 - Sets as unordered lists & as ordered lists(補足)
 - Sets as binary trees
 - Sets and information retrieval
- 2.4 Multiple Representations for Abstract Data
 - 2.4.2 Representations for Complex Numbers
 - 2.4.2 Tagged data
 - 2.4.3 Data-Directed Programming and Additivity
- 配布する用紙に名前を記入して下さい。
- 今週の宿題と共に提出して下さい。

30

複素数システムのデータ抽象化の壁

複素数を使ったプログラム
プログラム領域での複素数

add-complex, sub-complex, mul 等
複素数演算パッケージ

直交座標表現 (Rectangular representation)	極座標表現 (Polar representation)
---	---------------------------------

cons car cdr
リスト構造と基本マシン算術

31

複素数の演算

1. 虚数 (imaginary part)
 $z = x + iy \quad i^2 = -1$

2. 加算 (addition)
Real-part($z_1 + z_2$) = Real-part(z_1) + Real-part(z_2)
Imaginary-part($z_1 + z_2$) = Imaginary-part(z_1) + Imaginary-part(z_2)

3. 乗算 (multiplication)
Re($z_1 \cdot z_2$) = Re(z_1) · Re(z_2) - Im(z_1) · Im(z_2)
Im($z_1 \cdot z_2$) = Re(z_1) · Im(z_2) + Im(z_1) · Re(z_2)
Magnitude($z_1 \cdot z_2$) = Magnitude(z_1) · Magnitude(z_2)
Angle($z_1 \cdot z_2$) = Angle(z_1) + Angle(z_2)

複素数の四則演算 $z = x + iy = re^{i\theta}$

```
(define (add-complex z1 z2)
  (make-from-real-imag
   (+ (real-part z1) (real-part z2))
   (+ (imag-part z1) (imag-part z2)) ))
(define (sub-complex z1 z2)
  (make-from-real-imag
   (- (real-part z1) (real-part z2))
   (- (imag-part z1) (imag-part z2)) ))
(define (mul-complex z1 z2)
  (make-from-mag-ang
   (* (magnitude z1) (magnitude z2))
   (+ (angle z1) (angle z2)) ))
(define (div-complex z1 z2)
  (make-from-mag-ang
   (/ (magnitude z1) (magnitude z2))
   (- (angle z1) (angle z2)) ))
```

34

複素数の表現法

$z = x + iy = re^{iA}$

$r = \sqrt{x^2 + y^2}$
 $A = \arctan(y, x)$

$x = r \cos A$
 $y = r \sin A$

36

複素数の表現法の実装

$z = x + iy = re^{iA}$

```
(make-from-real-imag
 (real-part z) (imag-part z) )
```

```
(make-from-mag-ang
 (magnitude z) (angle z) )
```

$x = r \cos A$ $r = \sqrt{x^2 + y^2}$
 $y = r \sin A$ $A = \arctan(y, x)$

37

複素数の表現法

$z = x + iy = re^{iA}$

- 選択子(selectors)

```
(define (real-part z) (car z))
(define (imag-part z) (cdr z))
(define (magnitude z)
  (sqrt (+ (square (real-part z))
           (square (imag-part z)))))
(define (angle z)
  (atan (imag-part z) (real-part z)))
```
- 構築子(constructors)

```
(define (make-from-real-imag x y)
  (cons x y) )
(define (make-from-mag-ang r a)
  (cons (* r (cos a)) (* r (sin a))))
```

38

複素数の表現法(続) $z = x + iy = re^{iA}$

- 選択子(selectors)


```
(define (real-part z)
  (* (magnitude z) (cos (angle z))))
(define (imag-part z)
  (* (magnitude z) (sin (angle z))))
(define (magnitude z) (car z))
(define (angle z) (cdr z))
```
- 構築子(constructors)


```
(define (make-from-real-imag x y)
  (cons (sqrt (+ (square x) (square y)))
        (atan y x)))
(define (make-from-mag-ang r a)
  (cons r a))
```

39

図形言語に複素数を導入

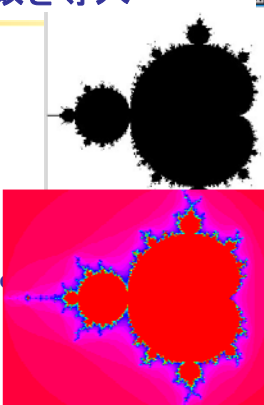
$$z_{n+1} = z_n^2 + C$$

$$z_0 = C$$

が収束する点 $C = (x, y)$
Mandelbrot Set

右上の図はframe coordinate
 map未使用(直接点を描画)

<http://mathworld.wolfram.com/MandelbrotSet.html>



12月13日・本日のメニュー

- 2.4 Multiple Representations for Abstract Data
- genetic procedures, type tags
 - 2.4.2 Representations for Complex Numbers
 - 2.4.2 Tagged data
 - 2.4.3 Data-Directed Programming and Additivity
- 配布する用紙に名前を記入して下さい。
- **今週の宿題と共に提出して下さい。**

41



2.4.2 Tagged data (タグ付きデータ)

- データ抽象化の1つの観点
- *Principle of least commitment* (最小責任の原則)
- 選択子と構築子を使用した抽象化の壁を使って、データオブジェクトの具体的な表現をできるだけ遅くし、システム設計における柔軟性を最大限にする。
- 本節ではさらに principle of least commitment を発展させる。

1. 表現法(選択子と構築子)の設計後でも、表現法の抽象化(曖昧性)を維持。
2. 直交座標と極座標が共用できる仕組みを考える。



タグ付きデータの実装法

- 手続きは `type-tag` (型タグ) で処理を区別する。
- `type-tag` はデータに付与されている。

```
(define (attach-tag type-tag contents)
  (cons type-tag contents))
(define (type-tag datum)
  (if (pair? datum)
      (car datum)
      (error "Bad tagged datum -
              TYPE-TAG" datum)))
(define (contents datum)
  (if (pair? datum)
      (cdr datum)
      (error "Bad tagged datum -
              CONTENTS" datum)))
```

47



座標のタグ付きデータの表現法

```
(define (rectangular? z)
  (eq? (type-tag z) 'rectangular))
(define (polar? z)
  (eq? (type-tag z) 'polar))
```

48

直交座標のタグ付きデータの表現法

```
(define (real-part-rectangular z) (car z))
(define (imag-part-rectangular z) (cdr z))
(define (magnitude-rectangular z)
  (sqrt (+ (square (real-part-rectangular z))
           (square (imag-part-rectangular z))
           )))
(define (angle-rectangular z)
  (atan (imag-part-rectangular z)
        (real-part-rectangular z) ))
(define (make-from-real-imag-rectangular x y)
  (attach-tag 'rectangular (cons x y) ))
(define (make-from-mag-ang-rectangular r a)
  (attach-tag 'rectangular
              (cons (* r (cos a)) (* r (sin a))) ))
```

50

極座標のタグ付きデータの表現法

```
(define (real-part-polar z)
  (* (magnitude-polar z)
     (cos (angle-polar z) )))
(define (imag-part-polar z)
  (* (magnitude-polar z)
     (sin (angle-polar z) )))
(define (magnitude-polar z) (car z))
(define (angle-polar z) (cdr z))
(define (make-from-real-imag-polar x y)
  (attach-tag 'polar
              (cons (sqrt (+ (square x) (square y)))
                    (atan y x) )))
(define (make-from-mag-ang-polar r a)
  (attach-tag 'polar (cons r a) ))
```

52

タグ付きデータへの手続き

```
(define (real-part z) dispatching on type
  (cond ((rectangular? z)
         (real-part-rectangular (contents z)))
        ((polar? z)
         (real-part-polar (contents z)))
        (else (error "Unknown type -
                      REAL-PART" z))))
(define (imag-part z)
  (cond ((rectangular? z)
         (imag-part-rectangular (contents z)))
        ((polar? z)
         (imag-part-polar (contents z)))
        (else (error "Unknown type -
                      IMAG-PART" z))))
```

53

タグ付きデータへの手続き(続)

dispatching on type

```

(define (magnitude z)
  (cond ((rectangular? z)
        (magnitude-rectangular (contents z)))
        ((polar? z)
         (magnitude-polar (contents z)))
        (else (error "Unknown type -
                      MAGNITUDE" z))))

(define (angle z)
  (cond ((rectangular? z)
        (angle-rectangular (contents z)))
        ((polar? z)
         (angle-polar (contents z)))
        (else (error "Unknown type -
                      ANGLE" z))))

```

54

タグ付きデータへの手続き(続々)

- 複素数の演算は**不変**
- 複素数汎用演算(generic operation)使用の為

```

(define (add-complex z1 z2)
  (make-from-real-imag
   (+ (real-part z1) (real-part z2))
   (+ (imag-part z1) (imag-part z2)) ))

(define (mul-complex z1 z2)
  (make-from-mag-ang
   (* (magnitude z1) (magnitude z2))
   (+ (angle z1) (angle z2))))

```

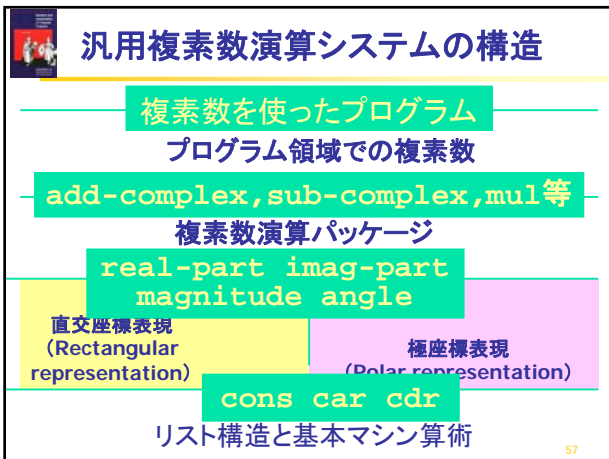
- 前と同じことに注意
- Principle of least commitmentによる**遅延**

55

全システム的设计は

- 目的に合致した複素数表現を選ぶ
- 直交座標表現
実数部と虚数部が分かっているとき
- 極座標表現
半径と角度が分かっているとき
- 最終的に得られた複素数演算システムの構造は次のスライド
- type-tag の使用がポイント
- *dispatching on type* という技法

56



汎用複素数演算システムの構造

- 複素数を使ったプログラム
 - プログラム領域での複素数
- add-complex, sub-complex, mul 等
 - 複素数演算パッケージ
- real-part imag-part magnitude angle
 - 直交座標表現 (Rectangular representation)
 - 極座標表現 (Polar representation)
- cons car cdr
 - リスト構造と基本マシン算術

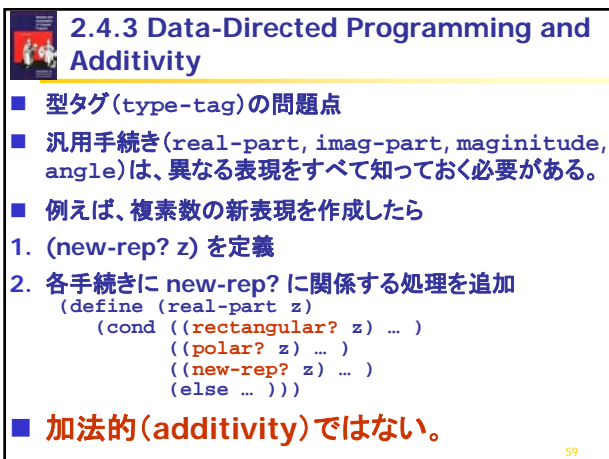
57



12月13日・本日のメニュー

- 2.4 Multiple Representations for Abstract Data
 - genetic procedures, type tags
 - 2.4.2 Representations for Complex Numbers
 - 2.4.2 Tagged data
 - 2.4.3 Data-Directed Programming and Additivity
 - 配布する用紙に名前を記入して下さい。
 - 今週の宿題と共に提出して下さい。

58



2.4.3 Data-Directed Programming and Additivity

- 型タグ (type-tag) の問題点
- 汎用手続き (real-part, imag-part, magnitude, angle) は、異なる表現をすべて知っておく必要がある。
- 例えば、複素数の新表現を作成したら
 - (new-rep? z) を定義
 - 各手続きに new-rep? に関する処理を追加


```
(define (real-part z)
  (cond ((rectangular? z) ... )
        ((polar? z) ... )
        ((new-rep? z) ... )
        (else ... )))
```
- 加法的 (additivity) ではない。

59

Data-Directed Programming (データ駆動型プログラミング)

- 加法的(additivity)なインターフェースとするために、表のようなデータを使用。

		型 (type)	
		Polar	Rectangular
演算 operations	real-part	real-part-polar	real-part-rectangular
	imag-part	imag-part-polar	imag-part-rectangular
	magnitude	magnitude-polar	magnitude-rectangular
	angle	angle-polar	angle-rectangular

60

表の操作

- 表に演算名・型(type)でその処理法をputで付加
- 表から演算名・型(type)でその処理法をgetで検索
- (put <op> <type> <item>)
表に<op> <type>で索引をつけて<item>を登録
- (get <op> <type>)
表から<op> <type>の索引で検索し、あれば、<item>を抽出
- TUT-Scheme(tus2, tustk2)では、
- (define put putprop)
- (define get getprop)

62

put と get の動き

```
(put 'banana 'price 300)
(put 'banana 'color ' yellow)
(get 'banana 'price)
(put 'Kyoto 'Ja "kyouto")
(put 'University 'Ja "daigaku")
(get 'Kyoto 'Ja)
  -> "kyouto"
(map (lambda (x) (get x 'Ja) )
     '(Kyoto University) )
  -> ("kyouto" "daigaku")
(put 'University 'Ge "Universitate")
```

63



簡単な情報検索 by put & get

```
(define (lookup given-key set-of-records)
  (let ((result (get set-of-records given-key))
        (if (null? result) false result)))
  (put 'population 'China '(1285.0 660.5 624.5))
  (put 'population 'India '(1025.1 528.5 496.6))
  (put 'population 'USA '(285.9 141.0 144.9))
  (put 'population 'Indonesia '(214.8 107.8
107.1))
  (put 'population 'Brazil '(172.6 85.2 87.4))
  (put 'population 'Pakistan '(145.0 74.5 70.5))
  (put 'population 'Russia '(144.7 67.7 77.0))
  (put 'population 'Bangladesh '(140.4 72.3 68.0))
  (put 'population 'Japan '(127.1 62.2 65.0))
  (put 'population 'Nigeria '(116.9 59.0 58.0))
  (put 'population 'Mexico '(100.4 49.6 50.7))
  (lookup 'Japan 'population)
```

64



表の操作で使うデータ

- 演算に関連する情報<item>は、以下では、**手続き(ラムダ式)**
- <type>は、**引数の型のリスト**
- 表に演算名・型(type)でその処理法をputで付加
- 表から演算名・型(type)でその処理法をgetで検索
- (put <op> <type> <item>)
表に<op> <type>で索引をつけて<item>を登録
- (get <op> <type>)
表から<op> <type>の索引で検索し、あれば、<item>を抽出

65



直交座標のタグ付きデータの表現法

```
(define (install-rectangular-package)
  ;; internal procedures
  (define (real-part z) (car z))
  (define (imag-part z) (cdr z))
  (define (make-from-real-imag x y)
    (cons x y))
  (define (magnitude z)
    (sqrt (+ (square (real-part z))
             (square (imag-part z)))))
  (define (angle z)
    (atan (imag-part z) (real-part z)))
  (define (make-from-mag-ang r a)
    (cons (* r (cos a)) (* r (sin a))))
  続く;; interface to the rest of the system
```

67



直交座標のタグ付きデータの表現法

```

(define (install-rectangular-package)
  ;; interface to the rest of the system
  (define (tag x)
    (attach-tag 'rectangular x) )
  (put 'real-part '(rectangular) real-part)
  (put 'imag-part '(rectangular) imag-part)
  (put 'magnitude '(rectangular) magnitude)
  (put 'angle '(rectangular) angle)
  (put 'make-from-real-imag 'rectangular
      (lambda (x y)
        (tag (make-from-real-imag x y))))
  (put 'make-from-mag-ang 'rectangular
      (lambda (r a)
        (tag (make-from-mag-ang r a))))
  'done)

```

68



直交座標のタグ付きデータの表現法

```

(define (install-rectangular-package)
  ;; internal procedures
  (define (real-part z) (car z))
  (define (imag-part z) (cdr z))
  (define (make-from-real-imag x y) (cons x y))
  (define (magnitude z)
    (sqrt (+ (square (real-part z))
             (square (imag-part z)))))
  (define (angle z)
    (atan (imag-part z) (real-part z)))
  (define (make-from-mag-ang r a)
    (cons (* r (cos a)) (* r (sin a))))
  ;; interface to the rest of the system
  (define (tag x) (attach-tag 'rectangular x))
  (put 'real-part '(rectangular) real-part)
  (put 'imag-part '(rectangular) imag-part)
  (put 'magnitude '(rectangular) magnitude)
  (put 'angle '(rectangular) angle)
  (put 'make-from-real-imag 'rectangular
      (lambda (x y) (tag (make-from-real-imag x y))))
  (put 'make-from-mag-ang 'rectangular
      (lambda (r a) (tag (make-from-mag-ang r a))))
  'done)

```

69



極座標のタグ付きデータの表現法

```

(define (install-polar-package)
  ;; internal procedures
  (define (magnitude z) (car z))
  (define (angle z) (cdr z))
  (define (make-from-mag-ang r a)
    (cons r a))
  (define (real-part z)
    (* (magnitude z) (cos (angle z))))
  (define (imag-part z)
    (* (magnitude z) (sin (angle z))))
  (define (make-from-real-imag x y)
    (cons (sqrt (+ (square x) (square y)))
          (atan y x)))

```

続く; interface to the rest of the system

70

極座標のタグ付きデータの表現法

```
(define (install-rectangular-package)
  ;; interface to the rest of the system
  (define (tag x) (attach-tag 'polar x))
  (put 'real-part '(polar) real-part)
  (put 'imag-part '(polar) imag-part)
  (put 'magnitude '(polar) magnitude)
  (put 'angle '(polar) angle)
  (put 'make-from-real-imag 'polar
      (lambda (x y)
        (tag (make-from-real-imag x y))))
  (put 'make-from-mag-ang 'polar
      (lambda (r a)
        (tag (make-from-mag-ang r a))))
  'done)
```

71

極座標のタグ付きデータの表現法

```
(define (install-polar-package)
  ;; internal procedures
  (define (magnitude z) (car z))
  (define (angle z) (cdr z))
  (define (make-from-mag-ang r a) (cons r a))
  (define (real-part z)
    (* (magnitude z) (cos (angle z))))
  (define (imag-part z)
    (* (magnitude z) (sin (angle z))))
  (define (make-from-real-imag x y)
    (cons (sqrt (+ (square x) (square y)))
          (atan y x)))
  ;; interface to the rest of the system
  (define (tag x) (attach-tag 'polar x))
  (put 'real-part '(polar) real-part)
  (put 'imag-part '(polar) imag-part)
  (put 'magnitude '(polar) magnitude)
  (put 'angle '(polar) angle)
  (put 'make-from-real-imag 'polar
      (lambda (x y) (tag (make-from-real-imag x y))))
  (put 'make-from-mag-ang 'polar
      (lambda (r a) (tag (make-from-mag-ang r a))))
  'done)
```


72

generic operation の適用方法

```
(define (apply-generic op . args)
  (let ((type-tags (map type-tag args)))
    (let ((proc (get op type-tags)))
      (if proc
          (apply proc (map contents args))
          (error
            "No method for these types -
            APPLY-GENERIC"
            (list op type-tags))))))
  'done)
```

これでgeneric procedure を再定義する。

```
(define (real-part z)
  (apply-generic 'real-part z))
```

 generic operation の適用方法

```

(define (real-part z)
  (apply-generic 'real-part z))


(define (imag-part z)
  (apply-generic 'imag-part z))

(define (magnitude z)
  (apply-generic 'magnitude z))

(define (angle z)
  (apply-generic 'angle z))

```

74

 目的に合致した複素数表現を選ぶ

- 直交座標表現 if 実数部と虚数部が分かっているとき
- 極座標表現 if 半径と角度が分かっているとき

```


(define (make-from-real-imag x y)
  ((get 'make-from-real-imag 'rectangular)
   x y))

(define (make-from-mag-ang r a)
  ((get 'make-from-mag-ang 'polar) r a))

```

- ほうら、うまく目的に合致した複素数表現が選ばれて、その後、型に対応した手続きが自動的に選択されることがお分かりになったでしょう。

75

 宿題: 配布用紙と共に提出すること

- 宿題は、次の計5問:
- Ex.2.63, 2.64, 2.65,
- Ex.2.73, 2.74

12月18日午後5時締切

DON'T PANIC!

