2.3 Multiple Definitions of `reverse` and `length` (複数の定義)

```
(define (reverse-rec items)
  (if (null? items)
      ()
      (append (list (car items))
              (reverse-rec (cdr items)) )))
```

```
(define (length-rec items)
  (if (null? items)
      0
      (+ 1 (length-rec (cdr items))) ))
```

```
(define (reverse-iter items)
  (define (iter results items)
    (if (null? items)
        results
        (iter (cons (car items) results)
              (cdr items) )))
  (iter () items) )
```

```
(define (length-iter items)
  (define (iter results items)
     (if (null? items)
         results
         (iter (+ 1 results)
               (cdr items) )))
   (iter 0 items) )
```

```
(define (reverse-r items)
  (fold-right
     (lambda (x r) (append r (list x)))
     () items ))
```

```
(define (length-r items)
  (fold-right
     (lambda (x y) (+ y 1))
     0 items ))
```

```
(define (reverse-l items)
  (fold-left
     (lambda (x y) (cons y x))
     () items ))
```

```
(define (length-l items)
  (fold-left
     (lambda (x y) (+ x 1))
     0 items ))
```

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

Generic procedures (汎用手続き), Type tag (型タグ), データ駆動型 (Data-directed),

2.4.1 Representations for Complex Numbers

1. **Rectangular Representation (直交座標系, real part and imaginary part, 実数部と虚数部)** $\Re(z_1 + z_2) = \Re(z_1) + \Re(z_2)$, $\Im(z_1 + z_2) = \Im(z_1) + \Im(z_2)$
2. **Polar Representation (magnitde and angle, 絶対値と偏角)** $\mathrm{Magnitude}(z_1 + z_2) = \mathrm{Magnitude}(z_1) * \mathrm{Magnitude}(z_2)$, $\mathrm{Angle}(z_1 + z_2) = \mathrm{Angle}(z_1) + \mathrm{Angle}(z_2)$

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

2.4.2 Tagged Data

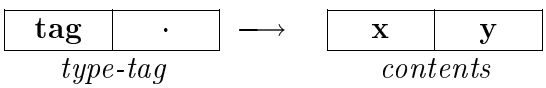
```
(define (attach-tag type-tag contents)
  (cons type-tag contents))
```

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

```
(define (type-tag datum)
  (if (pair? datum)
      (car datum)
      (error "Bad tagged datum -- TYPE-TAG" datum)))
```

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

```
(define (contents datum)
  (if (pair? datum)
      (cdr datum)
      (error "Bad tagged datum -- CONTENTS" datum)))
```



```
(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))) ))
```

```
(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)))
```

Generic Selectors

```
(define (real-part z)
  (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)) ))

(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)) ))

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

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

(define (make-from-mag-ang r a) (make-from-mag-ang-polar r a))
```

2.4.3 Deta-Directed Programming and Additivity (データ駆動型プログラミングと加法性)

- (put ⟨*op*⟩ ⟨*type*⟩ ⟨*item*⟩)
- (get ⟨*op*⟩ ⟨*type*⟩)
- *a package*: **a collection of procedures**

```
(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)
```

```
(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)
```

```
(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) )))))

;;  Generic selectors
(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))

;;  Constructors for complex numbers
(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))
```

2.4.4 Message Passing (メッセージパッシング)

```
(define (make-from-real-imag x y)
  (define (dispatch op)                          (define (apply-generic op arg) (arg op))
    (cond ((eq? op 'real-part) x)
          ((eq? op 'imag-part) y)
          ((eq? op 'magnitude)
           (sqrt (+ (square x) (square y))))
          ((eq? op 'angle) (atan y x))
          (else (error "Unknown op -- MAKE-FROM-REAL-IMAG" op))))
  dispatch )
```

3 宿題 – メ切: １月１１日 (火) 午前１２時 事務室レポート箱

- **Exercise 2.59 〜 2.76**
- **プログラムが動くことにより解を確認すること. (添削が試験までに間に合うかは?)**

4 練習問題 2.6 — Church numerals の加算と乗算

```
(define zero (lambda (f) (lambda (x) x)))
(define (add-1 n) (lambda (f) (lambda (x) (f ((n f) x)))))      (add-1 zero) ↓
(define one (lambda (f) (lambda (x) (f x))))   by substitution model for procedure application
(define two (lambda (f) (lambda (x) (f (f x)))))              ⇐ (add-1 one)
(define three (lambda (f) (lambda (x) (f (f (f x))))))        ⇐ (add-1 two)

(define (add n m) (lambda (f) (lambda (x) ((m f) ((n f) x)))))
(define (multiply n m) (lambda (f) (lambda (x) ((n (m f)) x))))

(define (numeral->number n)
  (define (successor x) (+ 1 x))
  ((n successor) 0) )

(define (equal n m)
   (= (numeral->number n) (numeral->number m)))
(define (greater n m)
   (> (numeral->number n) (numeral->number m)))

(define (subtract n m)
  (define (sub m result)
    (if (equal n m)
        result
        (sub (add-1 m) (add-1 result) )))
  (if (greater m n)
      zero
      (sub m zero) ))

(numeral->number (add three two))      (numeral->number (subtract three two))
(numeral->number (multiply three two)) (numeral->number (add (multiply three two) three))
```

5 自己参照文 (Self-Reference Sentence) by Douglas R. Hofstadter

- "Cette phrase en français est difficile à traduire en japonais." **の翻訳は.**
  「フランス語の文は日本語に翻訳するのが難しい. 」でよいか.
- This sentence contains ten words, eighteen syllables and sixty-four letters.
- "Has eighteen letters" does.
- Only the fool would take trouble to verify that this sentence was composed of ten a's, three b's, four c's, four d's, forty-six e's, sixteen f's, four g's, thirteen h's, fifteen i's, two k's, nine l's, four m's, twenty-five n's, twenty-four o's, five p's, sixteen r's, forty-one s's, thirty-seven t's, ten u's, eight v's, eight w's, four x's, eleven y's, twenty-seven commas, twenty-three apostrophes, seven hyphens and, last but not least, a single!
- 'T' is the first, fourth, eleventh, sixteenth, twnety-fourth, twenty-ninth, thirty-third, ···