「アルゴリズムとデータ構造入門」 第 9 回 資料 (*Dec. 7, 2004,* 奥乃)
http://winnie.kuis.kyoto-u.ac.jp/∼okuno/Lecture/04/IntroAlgDS/

随意課題２ (期限: ２００５年３月１５日 — 延長しました)

これはすごいという抽象化を使った Scheme プログラム.
例えば, 整数論, 群論, 組合せ論, 線形計画法, 古典力学, パズル解法, ゲーム, など.

第２. ２節のコード

2.2.1 Mapping over lists

```
(define (scale-list items factor)
  (if (null? items)
      nil
      (cons (* (car items) factor)
            (scale-list (cdr items) factor))))

(define (map proc items)
  (if (null? items)
      nil
      (cons (proc (car items))
            (map proc (cdr items)))))

(define (scale-list items factor)
  (map (lambda (x) (* x factor))
       items))
```

2.2.2 Hierarchical Structures

```
(define (count-leaves x)
  (cond ((null? x) 0)
        ((not (pair? x)) 1)
        (else (+ (count-leaves (car x))
                 (count-leaves (cdr x))))))
```

Mapping over trees

```
(define (scale-tree tree factor)
  (cond ((null? tree) nil)
        ((not (pair? tree)) (* tree factor))
        (else (cons (scale-tree (car tree) factor)
                    (scale-tree (cdr tree) factor)))))

(define (scale-tree tree factor)
  (map (lambda (sub-tree)
         (if (pair? sub-tree)
             (scale-tree sub-tree factor)
             (* sub-tree factor)))
       tree))
```

2.2.3 Conventional Inerfaces

```
(define (sum-odd-squares tree)
  (cond ((null? tree) 0)
        ((not (pair? tree))
         (if (odd? tree) (square tree) 0))
        (else (+ (sum-odd-squares (car tree))
                 (sum-odd-squares (cdr tree))))))
```

```
(define (even-fibs n)
  (define (next k)
    (if (> k n)
        nil
        (let ((f (fib k)))
          (if (even? f)
              (cons f (next (+ k 1)))
              (next (+ k 1))))))
  (next 0))
```

Sequence operations

```
(define (filter predicate sequence)
  (cond ((null? sequence) nil)
        ((predicate (car sequence))
         (cons (car sequence)
               (filter predicate (cdr sequence))))
        (else (filter predicate (cdr sequence)))))
```

```
(define (accumulate op initial sequence)
  (if (null? sequence)
      initial
      (op (car sequence)
          (accumulate op initial (cdr sequence)))))
```

例

```
(define (enumerate-interval low high)
  (if (> low high)
      nil
      (cons low (enumerate-interval (+ low 1) high))))
```

```
(define (enumerate-tree tree)
  (cond ((null? tree) nil)
        ((not (pair? tree)) (list tree))
        (else (append (enumerate-tree (car tree))
                      (enumerate-tree (cdr tree))))))
```

```
(define (sum-odd-squares tree)
  (accumulate +
              0
              (map square
                   (filter odd?
                           (enumerate-tree tree)))))
```

```
(define (even-fibs n)
  (accumulate cons
              nil
              (filter even?
                      (map fib
                           (enumerate-interval 0 n)))))

(define (list-fib-squares n)
  (accumulate cons
              nil
              (map square
                   (map fib
                        (enumerate-interval 0 n)))))

(define (product-of-squares-of-odd-elements sequence)
  (accumulate *
              1
              (map square
                   (filter odd? sequence))))

(define (salary-of-highest-paid-programmer records)
  (accumulate max
              0
              (map salary
                   (filter programmer? records))))
```

Nested mappings

```
(accumulate append
            nil
            (map (lambda (i)
                 (map (lambda (j) (list i j))
                      (enumerate-interval 1 (- i 1))))
                 (enumerate-interval 1 n)))

(define (flatmap proc seq)
  (accumulate append nil (map proc seq)))

(define (prime-sum? pair)
  (prime? (+ (car pair) (cadr pair))))

(define (make-pair-sum pair)
  (list (car pair) (cadr pair) (+ (car pair) (cadr pair))))

(define (prime-sum-pairs n)
  (map make-pair-sum
       (filter prime-sum?
               (flatmap
                (lambda (i)
                  (map (lambda (j) (list i j))
                       (enumerate-interval 1 (- i 1))))
                (enumerate-interval 1 n)))))
```

```
(define (permutations s)
  (if (null? s)                          ; empty set?
      (list nil)                         ; sequence containing empty set
      (flatmap (lambda (x)
                 (map (lambda (p) (cons x p))
                      (permutations (remove x s))))
               s)))

(define (remove item sequence)
  (filter (lambda (x) (not (= x item)))
          sequence))
```

3 Terminology

- **Horner's rule**
- **N-Queens**

4 宿題 – メ切は従来通り: １２月１３日 (月) 午後５時 事務室レポート箱

1. **Exercise 2.22 〜 2.43**
2. **プログラムは動くことを確認すること.**
3. **N-Queens の解の個数 (除く対称解) をリストアップせよ. (N=1..8 は必修, それ以上は随意)**

5 補足

deca (da)	deci (d)
hecto (h)	centi (c)
kiko (k)	milli (m)
mega (M)	micro (myu)
giga (G)	nano (n)
tera (T)	pico (p)
peta (P)	femto (f)
exa (E)	atto (a)
zetta (Z)	zepto (z)
yotta (Y)	yocto (y)

10^1	ten or decad	
10^2	hundred or hecatontad	
10^3	thousand or chiliad	2^{10}
10^4	myriad	
10^5	lac or lakh	
10^6	million	2^{20}
10^7	crore	
10^8	myriamyriad	
10^9	milliard	
10^9	billion	2^{30}
10^{12}	trillion	2^{40}
10^{15}	quadrillion	2^{50}
10^{33}	decillion	2^{110}
10^{63}	vigintillion	2^{210}
10^{303}	centillion	
10^{100}	googol	
10^{googol}	googolplex	
10^N	Nplex	
10^{-N}	Nminex	