プログラミング言語論第4回

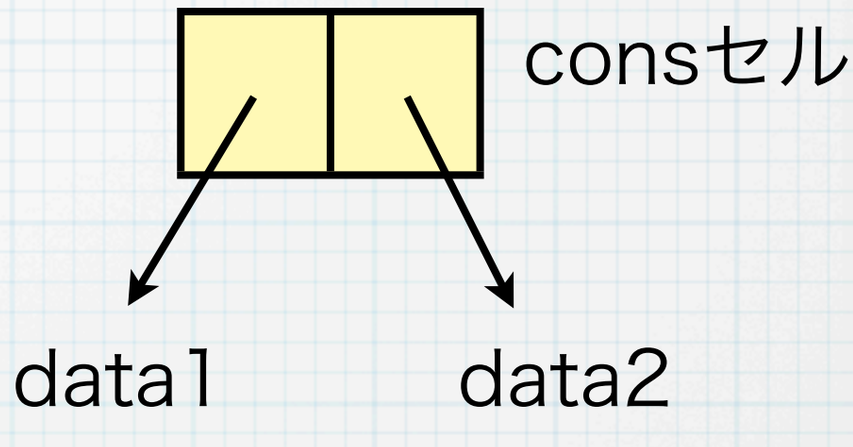
例題による解説（１）

情報工学科 山本修身

前回の復習 (1)

2つのデータを組み合せて1つのデータを作るためのデータとして consセルがある．consセルを作るための関数はconsである．

```
(cons data1 data2)
```



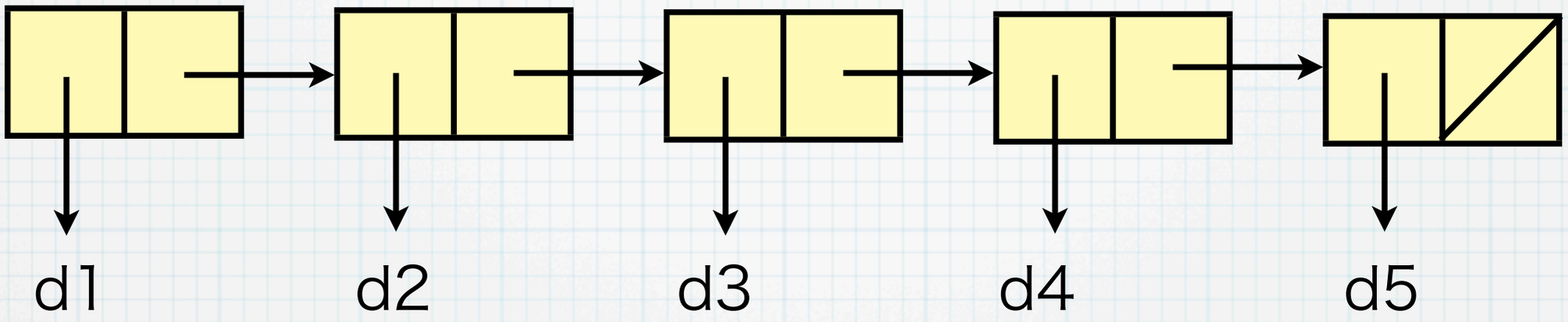
さらに，consセルのそれぞれ左側，右側のデータを取り出す関数として，carとcdrが用意されている．

```
(car pair)

(cdr pair)
```

```
p = (cons data1 data2)

    (car p) = data1
    (cdr p) = data2
```

前回の復習 (2)

consセルを連ねて，線形リストの構造を作ることができる.



線形リストを直接作る関数として list 関数がある.

```
(list d1 d2 d3 d4 d5)
        ||
(d1 d2 d3 d4 d5)
```

最後のセルの右側には何も入っていないが，これは通常( )と表現され，null値と呼ばれる．空のリストと考えても良い．nullであるか否かを判定する関数 null? が定義されている.

前回の復習(3)

与えられたS式を評価しないでそのまま値とするための特殊形式として，quoteが定義されている.

```
(quote S式)
```

通常，上記形式で書かなくても，'（クオート）をS式の先頭に置くことで同じ意味となる.

前回の復習 (4)

特殊形式 let によって，局所変数を定義し，letで定義された内部で式を評価することができる.

```
(let ((変数1 式1) ... (変数n 式n))
  本体の式1
  本体の式2
  ...
  本体の式m)
```

前回の復習 (5)

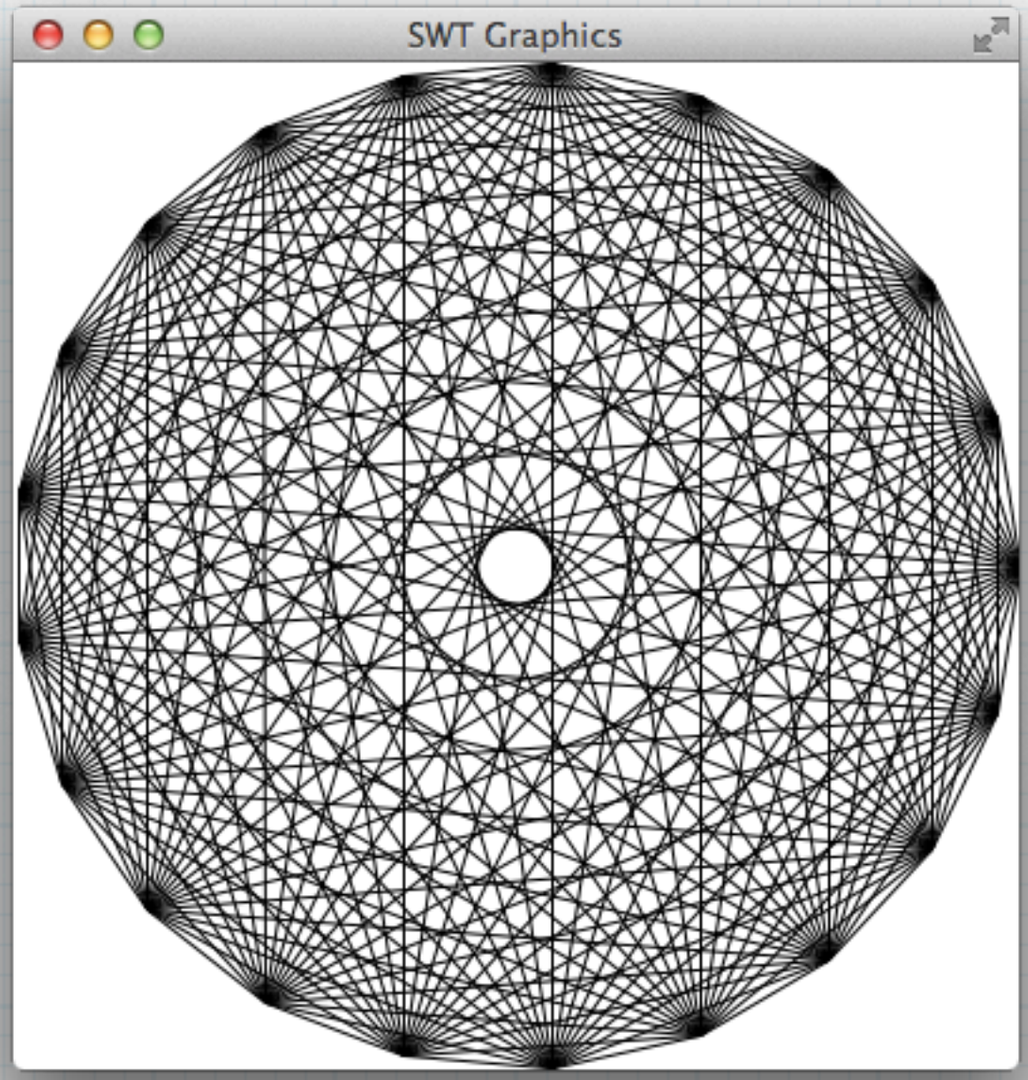
連想リストを定義し，それを用いることによって，色々な問題を可決することができる.

キー　値　キー　値

```
((key1 . val1) (key2 . val2) ...
               (keyn . valn))
```

例題１：図形描画

円周をn等分に分割したときのn点を互いに繋いでできる図形を表示せよ．表示には，SWT (Standard Widget Toolkit）をKawa処理系から呼べるようにしたものを用いよ.



例題1への解答例 (1)

SWTのインストール方法については，別資料を参照すること．山本が書いた graphics.scm を読み込んで，プログラムを書く．kawaのプログラムの主な構造は以下のようになる．

```
(load “graphics.scm”)
```
“graphics.scm” を読み込む

```
(define (draw gc disp)
   描画するためのプログラム)
```
gc はgraphics context の略で，これを用いて描画する

```
(graphics-env 400 400 draw)
```
↑ウィンドウの大きさ

例題1への解答例 (2)

まず，使う道具を定義する．nthは標準で定義されている場合が多いが（実際Schemeでは list-refという関数が定義されている），とりあえず，必要ではないかもしれないが，ここで定義する．さらに，繰り返しのための関数 myfor を定義する.

```
(define (nth n lst)
  (if (= n 0) (car lst)
      (nth (- n 1) (cdr lst))))
```

```
(define (myfor i n func)
  (if (= i n) '()
      (begin
        (func i)
        (myfor (+ i 1) n func))))
```

また，graphics context gcを用いて線分を描画するには，以下のようにする．点 (x1, y1)と(x2, y2)が結ばれる.

```
(gc:drawLine x1 y1 x2 y2)
```

例題1への解答例 (3)

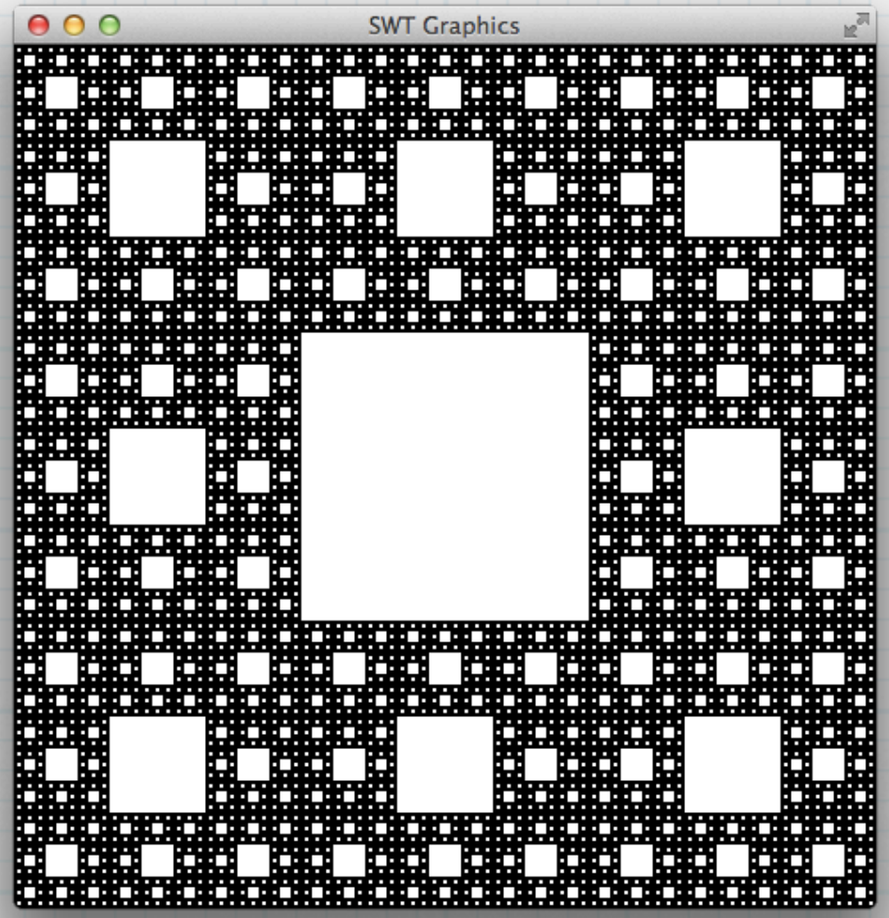
前のスライドで定義した関数を用いて，以下のようなプログラムを書く．最後にgraphics-envに渡して実行する．

```
(define (mandara gc disp)
  (let ((n 21)
        (pi 3.1415926)
        (r 200)
        (bx 200)
        (by 200))
    (define (make-points i res)
      (if (>= i n) res
          (let ((th (/ (* 2 pi i) n)))
            (let ((x (+ (* r (cos th)) bx))
                  (y (+ (* r (sin th)) by)))
              (make-points (+ i 1) (cons (cons x y) res))))))
    (let ((pts (make-points 0 '())))
      (myfor 0 n
             (lambda (i)
               (myfor (+ i 1) n
                      (lambda (j)
                        (let ((pt1 (nth i pts))
                              (pt2 (nth j pts)))
                          (gc:drawLine (car pt1) (cdr pt1)
                                       (car pt2) (cdr pt2)))))))))

(graphics-env 400 400 mandara)
```

←末尾再帰によって円周上の点の座標を計算してリストにする．

例題２： シェルピンスキーカーペット

グラフィックス環境を用いて，アルゴリズムデータ構造で出てきた
シェルピンスキーカーペットを描画せよ．



例題２への解答例 (1)

前問と同様にして，描画するプログラムを書けば良い．ただし，四角形を描画するには，以下の関数を用いる．

```
(gc:fillRectangle x y width height)
```

この関数で用いられる色は，背景色（バックグラウンド色）であり，デフォルトでは，白になっているので，これを黒に変える．そのための命令は以下のとおりである．

```
(gc:setBackground (disp:getSystemColor 2))
```

例題２への解答例 (2)

プログラムは以下のようになる．ただし，myforを用いている．

```
(load "graphics.scm")

(define winsize 486)

(define (sierpinski gc disp)
  (gc:setBackground (disp:getSystemColor 2))
  (define (sierpinski-iter size x y depth)
    (if (>= depth 5) (gc:fillRectangle x y size size)
        (let ((nsize (/ size 3))
              (ndepth (+ depth 1)))
          (myfor 0 3
                 (lambda (i)
                   (myfor 0 3
                          (lambda (j)
                            (if (not (and (= i 1) (= j 1)))
                                (sierpinski-iter nsize
                                                 (+ x (* i nsize))
                                                 (+ y (* j nsize))
                                                 ndepth))))))))))
  (sierpinski-iter winsize 0 0 0))

(graphics-env winsize winsize sierpinski)
```

例題３：万年カレンダー

西暦の年と月を指定して，その月のカレンダーを表示するプログラムを書け（これを万年カレンダーと呼ぶ）．太陽暦では，それぞれの年によって2月の日数を変える．通常２月は28日だが，うるう年（leap year）の２月は29日ある．うるう年は，西暦が4で割り切れる年とするが，例外として，100で割り切れる年はうるう年ではない．しかし，さらに例外として400で割り切れる年はうるう年である．この規則のみを用いて計算せよ．カレンダーの出力は以下のようにせよ．

```
    October  2015
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

例題３への解答例 (1)

まず，うるう年であるかどうかを判定する関数 leap-year? を作る.

```
(define (leapyear? year)
  (define (check n) (= (remainder year n) 0))
  (if (check 400) #t
      (if (check 100) #f
          (if (check 4) #t #f))))
```

この関数を用いて，西暦１年から与えられた年yearの前年までの日数を計算する.

```
(define (days-from-year-one year)
  (define (days-from-year-one-iter y year res)
    (if (= y year) res
        (days-from-year-one-iter
         (+ y 1)
         year
         (+ res (if (leapyear? y) 366 365)))))
  (days-from-year-one-iter 1 year 0))
```

例題３への解答例 (2)

さらに，月monthと年yearを指定して，西暦1年からその月の前日までの日数を計算する関数 total-days と以下のように定義する.

```
(define nmonth '(31 28 31 30 31 30 31 31 30 31 30 31))
(define (total-days month year)
  (define (total-days-iter m month total)
    (if (= m month) total
        (total-days-iter
         (+ m 1) month
            (+ total (nth (- m 1) nmonth)
               (if (and (= m 2) (leapyear? year)) 1 0)))))
  (total-days-iter 1 month (days-from-year-one year)))
```

関数 total-days を用いて，与えられた月の最初の日の曜日を計算する．0が日曜日，1が月曜日，…，6が土曜日と考える.

```
(define (day-of-week month year)
  (remainder (+ 1 (total-days month year)) 7))
```

例題３への解答例 (3)

カレンダーの日付を右づめ3桁で出力するために，与えられた数を右づめ3桁で出力する関数 int->string を定義する.

```
(define (int->string n)
  (if (< n 10) (string-append "  " (number->string n))
      (string-append " " (number->string n))))
```

さらに，空白を任意個続けて出力するために，n個の空白から構成される文字列を返す関数 spaces を以下のように定義する.

```
(define (spaces n)
  (if (= n 0) ""
      (string-append " " (spaces (- n 1)))))
```

月の名前と曜日の文字列を以下のように定義する.

```
(define monthname '("January" "February" "March" "April" "May"
                    "June" "July" "August" "September" "October"
                    "November" "December"))
(define weekname " Su Mo Tu We Th Fr Sa")
```

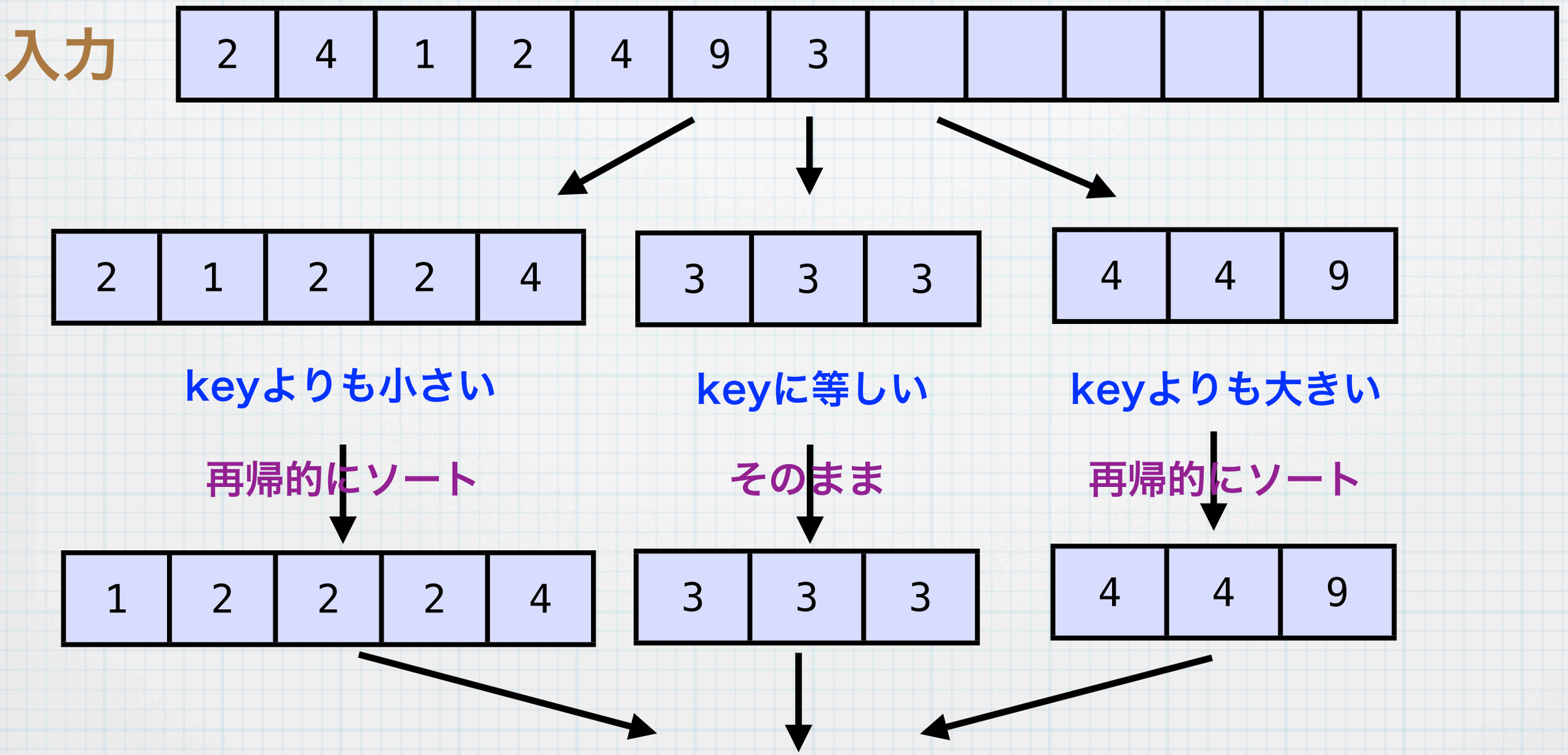
例題３への解答例 (4)

```
(define (prcal month year)
  (define (print-title)
    (let ((ss (string-append (nth (- month 1) monthname) " "
                             (int->string year))))
      (let ((ll (string-length ss)))
        (display (spaces (quotient (- 21 ll) 2)))
        (display ss)
        (newline))))
  (define (prcal-iter d md i)
    (if (> d md) '()
        (let ((ii (remainder (+ i 1) 7)))
          (display (int->string d))
          (if (=  ii 0) (newline))
          (prcal-iter (+ d 1) md ii))))
(print-title)
(display weekname)
(newline)
(let ((md (+ (nth (- month 1) nmonth)
             (if (and (= month 2) (leapyear? year)) 1 0)))
      (dw (day-of-week month year)))
  (display (spaces (* dw 3)))
  (prcal-iter 1 md dw)
  (newline)))
```

カレンダーの出力プログラムはこのようになる.

例題4：クイックソート

与えられた数値を要素とするリストをクイックソートアルゴリズムでソートしたリストを返すプログラムを書け.



例題４への解答例 (1)

プログラムをコンパクトに表現するためにいくつかの関数と特殊形式を導入する.

それぞれの条件が成り立つか上から順に調べて条件が真であれば，それに対応する式を評価して，その値をかえす．いずれも成り立たなければ，elseに対応する式 x を評価して返す.

```
(cond (条件１　式１)
      (条件２　式２)
       .......
      (条件ｎ　式ｎ)
      (else　 式ｘ))
```

リストの要素を取り出すためのいくつかの命令があらかじめ定義されている.

```
(cadr lst) = (car (cdr lst))
(caddr lst) = (car (cdr (cdr lst)))
(cadddr lst) = (car (cdr (cdr (cdr lst))))
```

例題４への解答例 (2)

プログラムはアルゴリズムをそのまま実現すれば良い.

```
(define (qsort lst)
  (define (qsort-filter lst key lt eq gt)
    (if (null? lst) (list lt eq gt)
        (let ((head (car lst))
              (rest (cdr lst)))
          (cond ((< head key) (qsort-filter rest key
                                            (cons head lt) eq gt))
                ((> head key) (qsort-filter rest key
                                            lt eq (cons head gt)))
                (else (qsort-filter rest key
                                    lt (cons head eq) gt))))))
  (if (<= (length lst) 1) lst
      (let ((key (car lst)))
        (let ((filtered (qsort-filter lst key '() '() '())))
          (let ((lt (car filtered))
                (eq (cadr filtered))
                (gt (caddr filtered)))
            (append (qsort lt) (append eq (qsort gt))))))))
```

例題４への解答例 (3)

ランダムでサイズの大きなリストをソートしてみる.

```
(define (make-random-list n res)
  (if (= n 0) res
      (make-random-list (- n 1)
                        (cons (java.lang.Math:random) res))))

(define (take n lst)
  (if (= n 0) '()
      (cons (car lst) (take (- n 1) (cdr lst)))))

(let ((t1 (java.lang.System:currentTimeMillis)))
  (let ((ans (qsort (make-random-list 100000 '()))))
    (let ((t2 (java.lang.System:currentTimeMillis)))
      (display (take 10 ans))
      (newline)
      (display (take 10 (reverse ans)))
      (newline)
      (display "Time = ") (display (- t2 t1)) (display " ms")
      (newline))))
```

乱数を発生させる関数

現在時間をミリ秒単位で返す関数

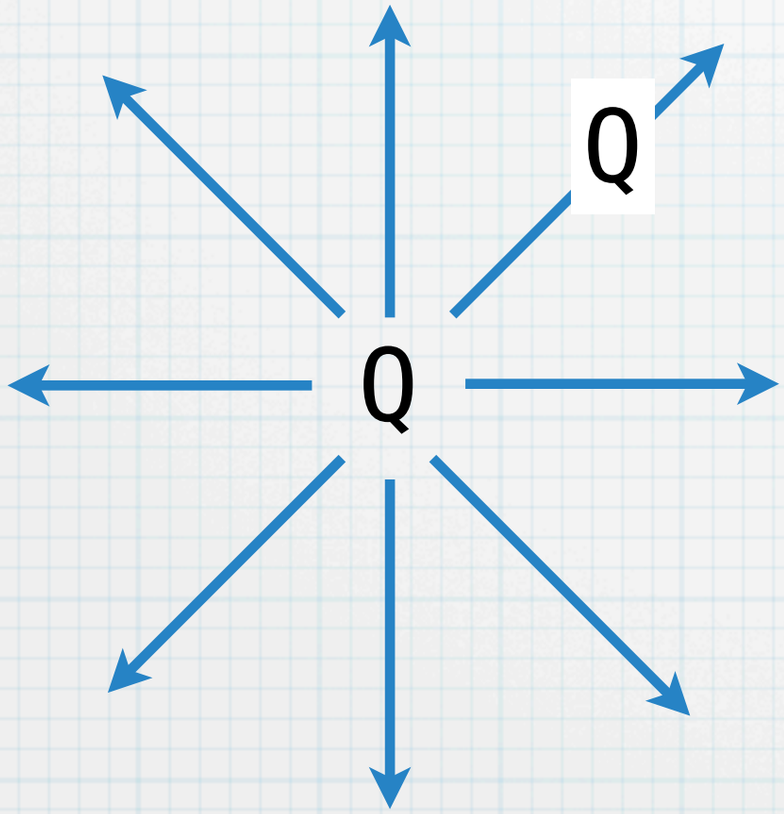
例題４への解答例 (4)

このプログラムを実際に動かすと以下のようになる.

```
OMacBook:yama535> kawa qsort.scm
(8.6465343406709561E-6 3.530463725298638E-5 3.5700397815863205E-5
 3.7146565792012254E-5 4.1509363431502244E-5 5.499105230000019E-5
 5.600812664552368E-5 7.183429281609754E-5 8.14970478996146 5E-5
 8.343786023112809E-5)
(0.9999953380659109 0.9999939515253108 0.999986356151066 1 0.9999803461073659
 0.9999741571916602 0.9999723067782 86 0.9999715824593057 0.9999667492422752
 0.9999519899539328 0.9999321690503455)
Time = 844 ms
OMacBook:yama536>
```

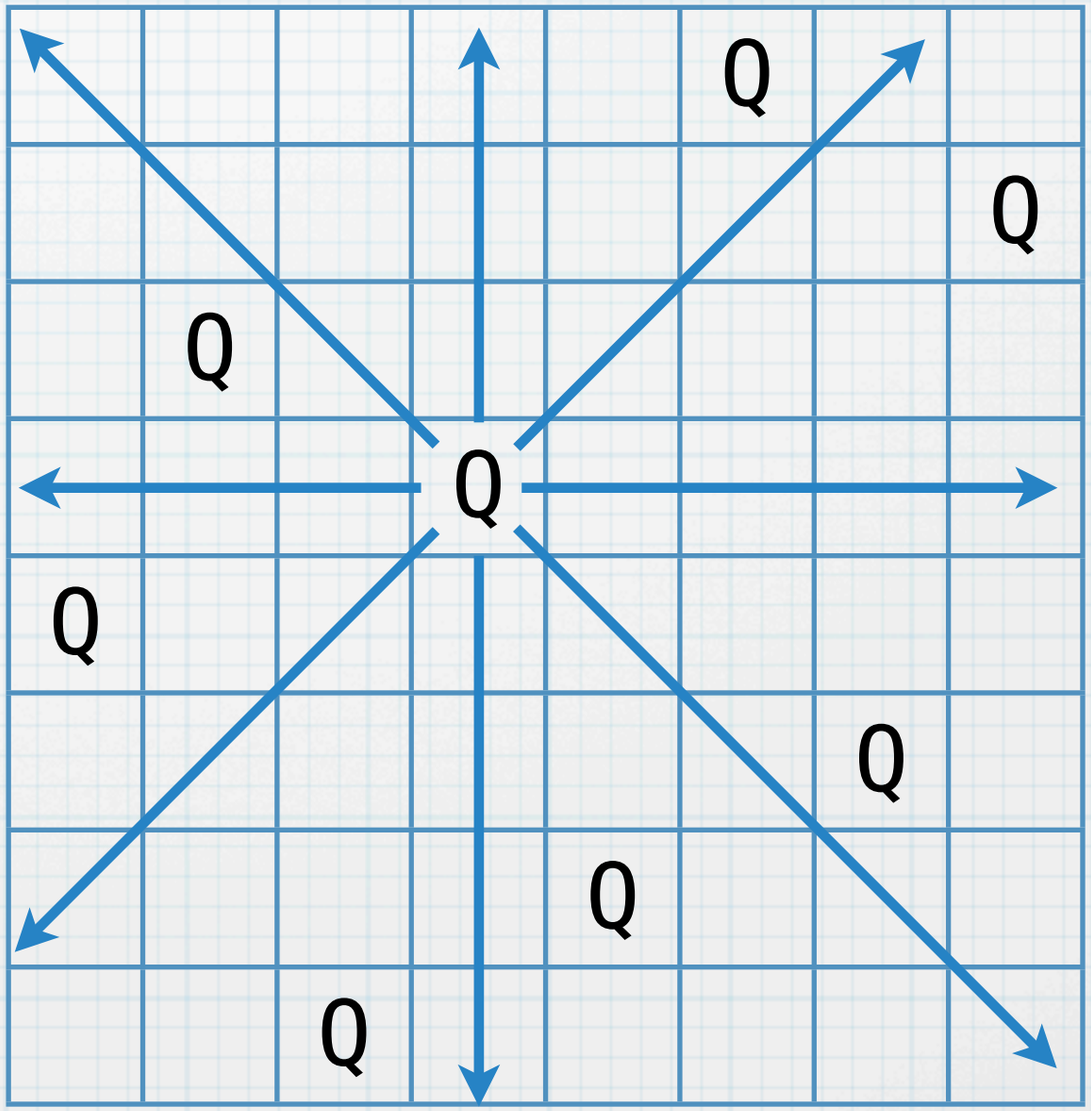
例題５

チェスゲームのためのボードは8x8に仕切られている．このボードに8つのクイーン (Q) を置いて，それぞれの Q が別の Q の効き筋におかれていないような配置をすべて求めよ（エイトクイーン問題）．



Qの効き筋

この問題で求める配置では
どのQもその効き筋に別のQ
が置かれていない



解の１つ

例題５への解答例 (1)

まず道具として2つの関数を用意する.

```
(define (myfor i n func)
  (if (>= i n) '()
      (begin
        (func i)
        (myfor (+ i 1) n func))))
```

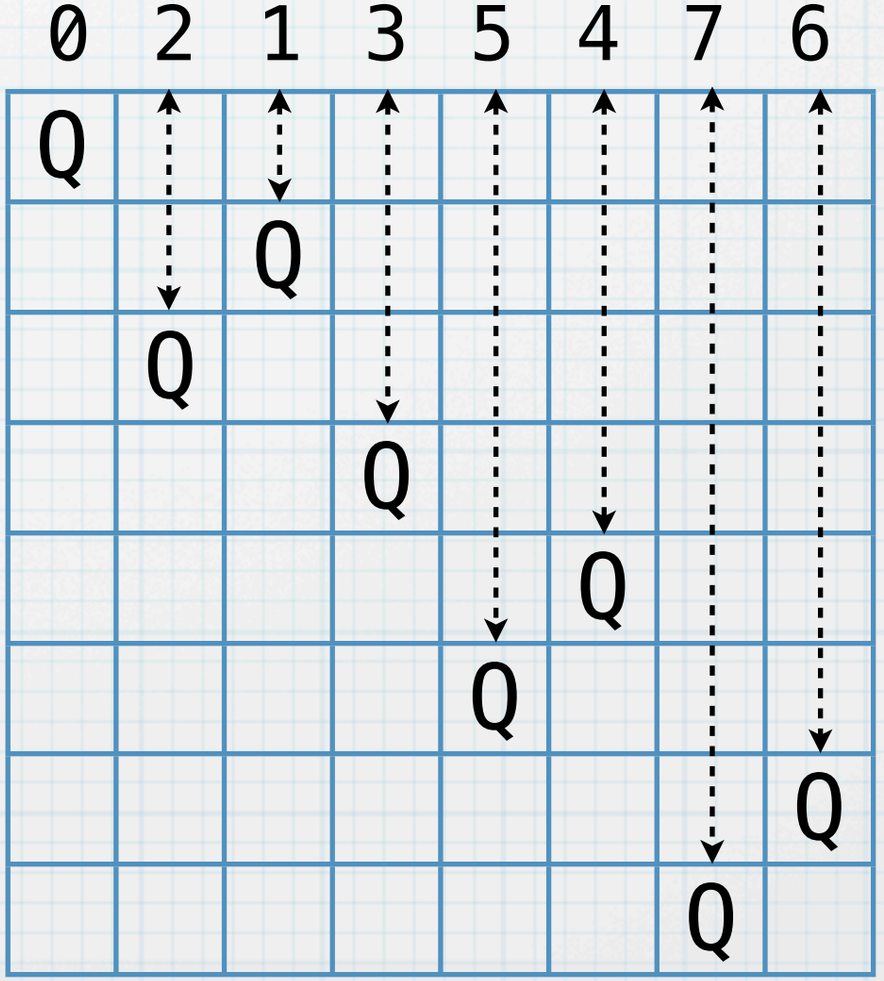
```
(define (contains x a)
  (cond  ((null? a) #f)
         ((= (car a) x) #t)
         (else (contains x (cdr a)))))
```

myfor は繰返しを実現する関数であり，contains は xがリストaの中に出現するかどうかを調べる関数である．contains は xがaに含まれるとき#fを，含まれないとき#tを返す.

例題５への解答例 (2)

解を表現するためにリストを用いる．ボードの大きさが8x8なので，それぞれの列に1つずつクイーンが並ぶ必要がある．そのため，表現リストでは，それぞれの列のどの行にクイーンが置かれているかということで表現する．たとえば (0 2 1 3 5 4 7 6)という列は，右下の図の配置を表している．

この問題の解はクイーンを並べながら順次リストを構成していく．部分的に出来上がった配置に新たに列をとある高さの Qで作ったとき，このQの効き筋にべつのQがあるかどうかの判定をする関数を作る.



例題５への解答例 (3)

高さiの列をlstに付け加えたとき，効き筋に当たるときは#f，当たらないときは#tと返す関数 check を以下のように定義する.

```
(define (check i lst)
  (define (check-iter i j lst)
    (if (null? lst) #t
        (let ((h (car lst)))
          (if (or (= (+ i j) h)
                  (= (- i j) h)
                  (= i h))
              #f
              (check-iter i (+ j 1) (cdr lst))))))
  (check-iter i 1 lst))
```

例題５への解答例 (4)

最終的な エイトクイーンの解を列挙する関数 eight-queen は以下のとおりである.

```
(define (eight-queen)
  (define (eight-queen-iter lst)
    (if (>= (length lst) 8)
        (begin (display lst)(newline))
        (myfor 0 8
               (lambda (i)
                 (if (and (not (contains i lst))
                          (check i lst))
                     (eight-queen-iter (cons i lst)))))))
  (eight-queen-iter '()))

(eight-queen)
```

例題５への解答例 (5)

答えは以下のようになる．92通り出力される．

```
(3 1 6 2 5 7 4 0)
(4 1 3 6 2 7 5 0)
(2 4 1 7 5 3 6 0)
(2 5 3 1 7 4 6 0)
(4 6 0 2 7 5 3 1)
(3 5 7 2 0 6 4 1)
(2 5 7 0 3 6 4 1)
(4 2 7 3 6 0 5 1)
(4 6 3 0 2 7 5 1)
(3 0 4 7 5 2 6 1)
(2 5 3 0 7 4 6 1)
(3 6 4 2 0 5 7 1)
(5 3 1 7 4 6 0 2)
(5 3 6 0 7 1 4 2)
(0 6 3 5 7 1 4 2)
(5 7 1 3 0 6 4 2)
(5 1 6 0 3 7 4 2)
(3 6 0 7 4 1 5 2)
(4 7 3 0 6 1 5 2)
(3 7 0 4 6 1 5 2)
(1 6 4 7 0 3 5 2)
(0 6 4 7 1 3 5 2)
(1 4 6 3 0 7 5 2)
(3 1 6 4 0 7 5 2)
(4 6 0 3 1 7 5 2)
(5 3 0 4 7 1 6 2)
(4 0 3 5 7 1 6 2)
(4 1 5 0 6 3 7 2)
(5 2 6 1 7 4 0 3)
(1 6 2 5 7 4 0 3)
(6 2 0 5 7 4 1 3)
(4 0 7 5 2 6 1 3)
(0 4 7 5 2 6 1 3)
(2 5 7 0 4 6 1 3)
(5 2 0 6 4 7 1 3)
(6 4 2 0 5 7 1 3)
(6 2 7 1 4 0 5 3)
(4 2 0 6 1 7 5 3)
(1 4 6 0 2 7 5 3)
(2 5 1 4 7 0 6 3)
(5 0 4 1 7 2 6 3)
(7 2 0 5 1 4 6 3)
(1 7 5 0 2 4 6 3)
(4 6 1 5 2 0 7 3)
(2 5 1 6 4 0 7 3)
(5 1 6 0 2 4 7 3)
(2 6 1 7 5 3 0 4)
(5 2 6 1 3 7 0 4)
(3 1 6 2 5 7 0 4)
(6 0 2 7 5 3 1 4)
(0 5 7 2 6 3 1 4)
(2 7 3 6 0 5 1 4)
(5 2 6 3 0 7 1 4)
(6 3 1 7 5 0 2 4)
(3 5 7 1 6 0 2 4)
(1 5 0 6 3 7 2 4)
(1 3 5 7 2 0 6 4)
(2 5 7 1 3 0 6 4)
(5 2 0 7 3 1 6 4)
(7 3 0 2 5 1 6 4)
(3 7 0 2 5 1 6 4)
(1 5 7 2 0 3 6 4)
(6 1 5 2 0 3 7 4)
(2 5 1 6 0 3 7 4)
(3 6 2 7 1 4 0 5)
(3 7 4 2 0 6 1 5)
(2 4 7 3 0 6 1 5)
(3 1 7 4 6 0 2 5)
(4 6 1 3 7 0 2 5)
(6 3 1 4 7 0 2 5)
(7 1 3 0 6 4 2 5)
(6 1 3 0 7 4 2 5)
(4 0 7 3 1 6 2 5)
(3 0 4 7 1 6 2 5)
(4 1 7 0 3 6 2 5)
(2 6 1 7 4 0 3 5)
(2 0 6 4 7 1 3 5)
(7 1 4 2 0 6 3 5)
(2 4 1 7 0 6 3 5)
(2 4 6 0 3 1 7 5)
(4 1 3 5 7 2 0 6)
(5 2 4 7 0 3 1 6)
(4 7 3 0 2 5 1 6)
(3 1 4 7 5 0 2 6)
(3 5 0 4 1 7 2 6)
(5 2 0 7 4 1 3 6)
(4 2 0 5 7 1 3 6)
(3 1 7 5 0 2 4 6)
(5 2 4 6 0 3 1 7)
(5 3 6 0 2 4 1 7)
(3 6 4 1 5 0 2 7)
(4 6 1 5 2 0 3 7)
```