■ 連載

Haskell プログラミング

暦法算法

和田 英一（IIJ 技術研究所）
wada@u-tokyo.ac.jp

新しき年の始めに

新(あらた)しき年の始に思ふどちい群れてをれば嬉しくもあるか	万葉集 4284
新(あらた)しき年の始の初春の今日ふる雪のいや重(し)け吉事(よごと)	万葉集 4516
年の内に春はきにけりひととせをこぞとはいはんことしとやいはん	古今和歌集 1

これらの和歌から古人の新年への特別の意識が推察できる．新年と分かるのは暦があったからだ．古来多くの暦法が工夫され，その算法が開発されてきた．今回はそれらの算法を Haskell 風に書く．

暦法算法の集大成が Reingold と Dershowitz の“Calendrical Calculations”[1]（以下「暦法算法の本」）で，各種の暦法の由来とその（本書独特の記法による）アルゴリズムが書いてある．独自記法のアルゴリズムも巻末（と CD）には Common Lisp による記述があり，私にはその方が理解しやすい．Knuth は The Art of Computer Programming（以下「TAOCP」）の 1 巻第 3 版で，日付けに関連するあらゆる種類のアルゴリズムが同書に詳しいという（問題 1.3.2-14 の解答）．

日本語では関係する法律の引用あり，専門用語の英語ありで，「時と暦」[2] が大変参考になる．かつて数学セミナーに掲載された島内剛一先生の「万年七曜表」[3] は暦法算法に関する蘊蓄に溢れていた．その他検索エンジンからさまざまな情報が得られる．

Unix では cal というカレンダーのコマンドが重宝で，引数を 0 個，1 個あるいは 2 個とり，0 個ならその月の，1 個なら引数で指定した年の，2 個なら指定した年と月のカレンダーを出力する．

```
% cal 1 2006
    January 2006
 S  M Tu  W Th  F  S
 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
```

```
% cal 9 1752
    September 1752
 S  M Tu  W Th  F  S
          1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

右側 1752 年 9 月のは英国（とアメリカ）が Gregorian 暦を採用した月のものだ．カトリックの国はローマ法王の改暦教書直後の 1582 年 10 月に（4 日の翌日を 15 日にして），日本は明治 6 年 1 月に（明治 6 年は旧暦で 13 カ月あり，月給を 1 カ月分節約すべく 5 年 12 月 2 日の翌日を 6 年 1 月にして[2]）採用した．

cal のような Gregorian 暦のプログラムを Haskell で書いてみよう．Richard Bird の本[4] にも“Print a Calendar”と

いう節があり，それも参考になる．ところで「ひとはなぜ山に登るのか」「それがそこにあるから」「ひとはなぜプログラムを書くのか」「それが楽しいから」．皆さんも楽しんでください．

Zeller の合同式

Gregorian 暦の西暦 y 年 m 月 d 日からその日の曜日を得るには Zeller の合同式が便利だ．「暦法算法の本」によると Christian Zeller が 1830 年代に考えたらしい．

m が 1 と 2 なら $m = m + 10; y = y - 1$（前年の 11 月と 12 月），そうでないなら $m = m - 2$（同年の 1 月から 10 月）とする．次に $a = y \text{ div } 100; b = y \bmod 100$ として $\{\lfloor 2.6m - 0.2 \rfloor + d + b + \lfloor \frac{b}{4} \rfloor + \lfloor \frac{a}{4} \rfloor - 2a\} \bmod 7$ を計算し，結果の値の 0 から 6 がこの順に日曜から土曜に対応する．

Zeller の合同式の味噌は 2 月を最後に回してうるう年の計算を省略した点にある．また $\lfloor 2.6m - 0.2 \rfloor$ の微妙な調整にある．**図 -1** がそれを示す．斜め線が $(2.6m - 0.2) \bmod 7$ で，$m = 1, 2, \ldots, 12$ との交点を黒まるで表す．左の方の 3/1 のすぐ上の白まるはある年の 3 月 1 日の曜日である．3 月 29 日も同じ曜日なので，その右に示す．その後，30 日，31 日，4 月 1 日の曜日も示す．またその右に 4 月 29 日を示し，5 月 1 日まで記入．これを繰り返してみると，各月の 1 日は斜め線の交点の最大整数部分に一致している．考えてみれば 3 月から 7 月の各月の日数は 31, 30, 31, 30, 31 日 ; 8 月から 12 月も同じ，さらに 1 月も 31 から始まる．この 5 カ月の各月の 4 週 (28 日) から余る日数の和は 3 + 2 + 3 + 2 + 3 = 13 日で，1 カ月に平均すると傾斜の 2.6 が得られる．では 0.2 は何だろう．

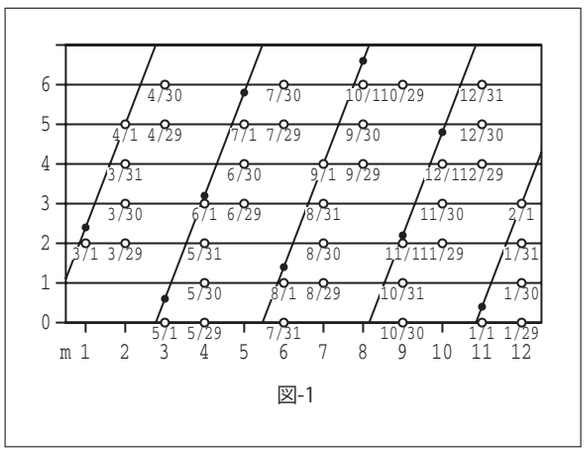

図-1

Haskell には `floor` という関数があるから，さっそく

```
zeller :: Int -> Int -> Int -> Int
zeller y m d = if m < 3 then z (y - 1) (m + 10) else z y (m - 2)
  where z y' m' = (floor(m' * 2.6 - 0.2) + d + b
         + floor(b / 4) + floor(a / 4) + 5 * a) `mod` 7
             where (a, b) = y' `divMod` 100
```

とやってみると見事失敗する．

```
ERROR "zeller.hs":2 - Instance of RealFrac Int required for definition of zeller
```

そこで 1 行目の型宣言をコメントアウトして読み込み，`zeller` の型を処理系に教えてもらう．

```
Main> :t zeller
zeller :: (RealFrac a, RealFrac b, Integral b) => b -> a -> b -> b
```

つまり `m` が RealFrac 型で，`y`, `d`, `zeller` が RealFrac，Integral 型といわれた．これは Haskell 独特の型推論の結果である．そもそもの原因は `floor` にあった．`floor` の型を聞くと

```
Main> :t floor
floor :: (RealFrac a, Integral b) => a -> b
```

すなわち `floor` の被演算子は RealFrac のインスタンスの型，Float か Double でなければならず，`m' * 2.6 - 0.2`

やｂ / ４やａ / ４も RealFrac であり，それらの被演算子 m', b, a も RealFrac であり，+ ｂや + 5 * ａを含む mod の左辺も RealFrac ということになる．一方 mod や div の型は

```
Main> :t div
div :: Integral a => a -> a -> a
```

したがって mod の左辺は Integral，それゆえｄも y' も ｙも Integral．結局ｍは RealFrac で，ｙとｄと zeller の結果は Integral と RealFrac との両方のインスタンスである型と推論される．そういう型はない．

この推論を止めるには，m', a, b を fromIntegral で保護する☆1．

```
zeller y m d = if m < 3 then z (y - 1) (m + 10) else z y (m - 2)
  where z y' m' = (floor(fromIntegral m' * 2.6 - 0.2) + d + b
         + floor(fromIntegral b / 4) + floor(fromIntegral a / 4) + 5 * a) `mod` 7
             where (a, b) = y' `divMod` 100
```

または floor ではなく div を使う．

```
  where z y' m' = ((m' * 26 - 2) `div` 10 + d + b + b `div` 4 + a `div` 4 + 5 * a)
                  `mod` 7
             where (a, b) = y' `divMod` 100
```

これでうまく行く．参考までに島内流では Gregorian 暦の西暦 *y* 年 *m* 月 *d* 日の曜日を

```
f, g, h :: Int -> Float
h m = [6.75,2.75,3.25,6.25,1.25,4.25,6.25,2.25,5.25,0.25,3.25,5.25]!!(m - 1)
g 0 = -0.25
g b = fromIntegral (b + c) - if d == 0 then 0.5 else 0
          where (c, d) = b `divMod` 4
f a = [5.875,4.125,2.125,0.125]!!(a `mod` 4)
dayOfWeek :: Int -> Int -> Int -> Int
dayOfWeek y m d = (round (f a + g b + h m) + d) `mod` 7
                     where (a, b) = y `divMod` 100
```

と計算する [3]．これら秘妙な数値から島内さんのハッカーぶりが彷彿とする．

Julian Date

天文学者は Julian Date (ユリウス日 JD) を使う．この Julian は Julius Cæsar とは関係ない．遥か昔まで Julian 暦だったと仮定し −4712 年 (紀元前 4713 年) 1 月 1 日 (月曜) 正午 UT (世界時) から始まる日を 0 とする通日で，表や計算式が理科年表や天文年鑑にある．1 日 (12 時〜 36 時) の起点が正午なのは天文学者の夜行性に基づく．これから (JD + 1) mod 7 で曜日が分かる．2006 年 1 月 1 日正午 UT の JD は**表 -1** から 2006 年 1 月 0 日の JD を得，それに 1 日の 1 を加えて 2453737 となる．この日は (2453737 + 1) mod 7 = 0 つまり日曜だ．1 月でなければ年初から前月末までの日数 (下のプログラムの mon0, mon1) を足す．

年	JD	年	JD	年	JD	年	JD	年	JD
2000	2451544	2002	2452275	2004	2453005	2006	2453736	2008	2454466
2001	2451910	2003	2452640	2005	2453371	2007	2454101	2009	2454832

表-1　1月0日正午UTのJD

Julian Date は理科年表などの表なしでもプログラムで計算できる．たとえば

☆1　これらのプログラムは http://www.sampou.org/haskell/ipsj/ から取ることができる．

```
mon0, mon1 :: [Int]          -- 年初から前月末日までの日数
mon0 = [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334]    -- 平年
mon1 = [0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335]    -- うるう年

gleap, jleap :: Int -> Bool          -- Gregorian, Julian 暦のうるう年に真を返す
gleap y = if y `mod` 100 == 0 then y `mod` 400 == 0 else y `mod` 4 == 0
jleap y = y `mod` 4 == 0

julianDate :: Int -> Int -> Int -> Int
julianDate y m d =                    -- a から g は途中の値
 let a = (y + 4712) * 365
     b = (y + 4712 + 3) `div` 4
     c = if y > 1601 then y' `div` 400 - y' `div` 100 else 0
           where y' = y - 1601
     e = if [y,m,d] >= [1582,10,15] then -10 else 0
     f = (if leap y then mon1 else mon0) !! (m - 1)
           where leap = if y > 1600 then gleap else jleap
     g = d - 1
  in a + b + c + e + f + g            -- 最後の結果
```

とする．簡単だが一応説明すると mon0，mon1 は第 0 要素が 1 月に対応し，その月の 0 日 (= 前月の末日) の年初からの日数である．mon1 はうるう年用．自分で加算するのが面倒なら

```
scanl (+) 0 [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30] --11 月までの各月の日数
```

で作れる．

gleap，jleap はそれぞれ Gregorian 暦と Julian 暦でうるう年に真を返す．Julian 暦は Julius Cæsar(−101−−43) が −45 年に採用したのだが，この式が紀元前にも当てはまるのは一見不思議である．「今年は −4 年でうるう年だ．そろそろ Y0K 年の準備をしよう」と古代人がいったとは思えない．前述の仮定のためである (当時はうるう年を誤解して 3 年に 1 度入れていた．それに気づき，しばらくうるうを中止し，西暦 8 年から正常に入れ始めたという説がある)．julianDate は西暦年 *y* 年 *m* 月 *d* 日を引数とし，JD を返す．一気に加算してもよいのだが途中結果 a，b，c，e，f，g は説明用だ．a はすべて平年とした場合の前年末までの総日数．b はそれまでの 4 で割れる年の総数．3 を足して 4 で割るのは大きい方に丸める常套手段である．Gregorian 暦は 1582 年から採用されたが，100 年目の補正が生じるのは 1700 年が最初である．c は 1601 年以降について，100 年，400 年の補正をする項．b で 4 で割れる年を全部足したのは足しすぎなので，100 で割れる年を引く．しかし 400 で割れる年も引いたのは引きすぎなので 400 で割れる年を足す．なんでもないことだが inclusion and exclusion principle というもっともらしい名前があり，「TAOCP」にはよく登場する．1582 年の改暦で飛ばした 10 日を減ずるのが e．リストの比較を利用した．f は mon でその年内で前月までの日数を得る．最後の -1 は 1 月 1 日の JD が 0 なことによる (julianDate (-4712) 1 1 => 0)．

カレンダーの表示

月初めの曜日を知る準備ができたので，最初に例示したような Gregorian 暦の月単位のカレンダーが書けるようになり，以下のプログラムを書いた．cal 2006 1 で本稿冒頭の例示と同様な 2006 年 1 月のカレンダーが得られる (Unix の cal と引数の順が逆なのに注意)．

```
intToString :: Int -> String -- カレンダーの日付けを 2 桁の文字列にする
intToString n = [" 123"!!a, ['0'..'9']!!b] where (a, b) = divMod n 10
```

```
intsToString :: [Int] -> Int -> String  --1 行 (7 日分 ) を文字列にする
intsToString ns ld = concatMap d ns     --concatMap は map して append
        where d x |x <= 0 || x > ld = "   "
                  |otherwise = intToString x ++ " "

monthnames :: [String]                  -- 見出しに使う月の名前
monthnames =["January","February","March","April","May","June","July",
             "August","September","October","November","December"]

month :: Int -> String          --21 文字に展開した月の名前
month m = expand (monthnames !! (m - 1))

expand :: String -> String      -- 文字列 s の両側に空白を置き 21 文字に展開する
expand s = let leng = length s  -- 文字列 s 自身の長さ
               padlen = (20 - leng) `div` 2          --片側の空白の文字数
           in take 21 (replicate padlen ' ' ++ s ++ repeat ' ')

cal :: Int -> Int -> IO ()      --y 年 m 月のカレンダーを出力
cal y m = do putStrLn head      -- 見出し ( 月  年 ) を出力
             putStrLn (unlines (cc y m))
               where head = expand ((monthnames !! (m - 1)) ++ " " ++ year)
                     year = show y          -- 西暦年 y を文字列に変換

daynames :: String
daynames = " S  M Tu  W Th  F  S "
leap :: Int -> Int              --y がうるう年なら 1，平年なら 0 を返す
leap y = dif 4 - dif 100 + dif 400
   where dif d = div y d - div y1 d; y1 = y - 1
cc :: Int -> Int -> [String]    --y 年 m 月のカレンダーの文字列を構成
cc y m = let z = 1 - zeller y m 1
             ld = [31,28+leap y,31,30,31,30,31,31,30,31,30,31]!!(m-1)
     in daynames : map (\x-> intsToString x ld) [[d..d+6]|d<-[z,z+7..z+35]]

-- 使い方 cal 2006 1
```

最初の intToString は引数 n を 2 桁の文字列に変換する．n が < 10 なら前に空白を置く．要するに 1 の桁と 10 の桁に分け，10 の桁は文字列 " ␣123"(空白␣に注意) から，1 の桁は文字列 "0123456789" (つまり文字のリスト) から，必要な文字を取り出しリストに構成する．日付けなので 31 日までできればよいとした．

intsToString は [-1,0,1,2,3,4,5] のような 7 個のリストと ld(last day) を貰い，0 以下および ld 超は空白としてカレンダーの 1 列を作る．この例だと " ␣␣␣␣␣␣␣1␣␣2␣␣3␣␣4␣␣5␣" ができる．利用している concatMap は ns に d を map し，++(append) を foldr1 するものである．

cal y m が *y* 年 *m* 月のカレンダーを出力する関数である．見出し head を出力し，下請け cc を y, m を引数で呼ぶ．cc は当月のカレンダーの各行のリストを返すので，それを改行で繋ぐ unlines で 1 本化し，出力する．年の文字列を作るには引数の整数を文字列化したものを返す show を利用する．

cc は Zeller の合同式で 1 日の曜日を得，それから最初の行を先頭へ (日曜へ) 外挿した日付け z を用意する．1 日が日曜なら 1，火曜なら 0，土曜なら −6 である．次に [[d..d+6]|d<-[z,z+7..z+35]] で z から始まる 6 週間分の日付けのリストを作り，それに intsToString を map する．

ld のために月毎の日数のリストがある．2 月はうるう年に 1 を返す関数 leap で補正する．

leap は**図 -2** に示すように y div 4 と $(y-1)$ div 4 の階段関数を作り，差から同図最下段の如く 4 で割れる年だけ 1 を返す．100 と 400 についても行えば，Gregorian 暦のうるう年に 1 を返すようなる．

1 年分のカレンダー

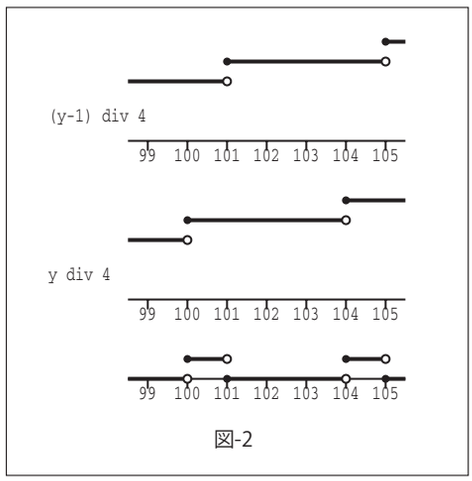

図-2

Unix の cal は引数の個数で仕事が変わるが，Haskell では引数の個数を変更が煩わしいから，呼出しの関数名を変えて 1 年分の方は cals y とした．

```
cals :: Int -> IO ()              --y 年のカレンダーを出力
cals y = do putStrLn (replicate 30 ' ' ++ show y ++ "\n")
                                  -- 見出しの出力
            putStrLn (unlines (map unwords mcc))
             where mcc = concatMap mc3 [1,4,7,10]
                   mc3 m = map f (zip3 (mc m) (mc (m + 1)) (mc (m + 2)))
                      where f (a, b, c) = [a, b, c]
                   mc m = month m : cc y m
-- 使い方 cals 2006
```

cals はまず出力する年で見出しを作り出力する．次に mc 1 は

```
["      January         "," S  M Tu  W Th  F  S "," 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             ","                     "]
```

を用意する．これを 2 月，3 月も作り，zip3 で 1 行目ごと，2 行目ごと，... にまとめる．zip3 は 3 つ組を返すので，f でリストへと構造変換する．それが mc3 の結果でこれを各四半期について作り，unwords を map し全体を unlines して出力する．

復活祭公式

暦法算法の重要な話題は復活祭公式である．そもそも改暦の理由の 1 つが復活祭であった．うるう年を入れすぎたため，16 世紀には春分が 3 月 11 日頃になり，改暦で 10 日を省いて調整した．復活祭は春分の日かその後の満月の後の日曜と決まっているが，春分は天文学的春分ではなく，3 月 21 日である．「TAOCP」1 巻の問題 1.3.2-32 (1 分冊では 1,3,2'-32) に復活祭公式がある．これは以下のような Kunth 流，つまり英語による手続き的な記述だ．

アルゴリズム E (復活祭の日取り) Y を復活祭の日を知りたい年とする．

E1. $G \leftarrow (Y \bmod 19) + 1$ とする．

E2. $C \leftarrow \lfloor Y/100 \rfloor + 1$ とする．

E3. $X \leftarrow \lfloor 3C/4 \rfloor - 12, Z \leftarrow \lfloor (8C + 5)/25 \rfloor - 5$ とする．

E4. $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$ とする．

E5. $E \leftarrow (11G + 20 + Z - \mathrm{X}) \bmod 30$ とする．

$E = 25$ で G が 11 を超えているか $E = 24$ なら E を 1 増やす．

E6. $N \leftarrow 44 - E$ とする．$N < 21$ なら $N \leftarrow N + 30$ とする．

E7. $N \leftarrow N + 7 - ((D + N) \bmod 7)$ とする．

E8. $N > 31$ なら日付けは $(N - 31)$ april, そうでないなら N march.

これを Haskell で関数風に書き直す．このアルゴリズムのようにデータ構造がないと，Haskell でも手続き型と似た構造になる（なお，第 47 回プログラミング・シンポジウム報告集まえがきも参照）．

```
import Time                              -- Module Time の Month 型を使うため
easter :: Int -> (Month, Int)
easter y =
 if n''>31 then (April,n''-31) else (March,n'')     -- E8
  where
    n''= n' + 7 - (d + n') `mod` 7                  -- E7
      where n'= if n < 21 then n + 30 else n        -- E6
            n = 44 - e'
    e'= if e == 25 && g > 11 || e == 24 then e + 1 else e   -- E5
      where e = (11 * g + 20 + z - x) `mod` 30
    d = (5 * y) `div` 4 - x - 10                    -- E4
    x = 3 * c `div` 4 - 12                          -- E3
    z = (8 * c + 5) `div` 25 - 5    -- `div`,`mod` は *,/ と同じ優先度
    c = y `div` 100 + 1                             -- E2
    g = y `mod` 19 + 1                              -- E1
```

ここで g は黄金数 (golden number). 太陽と月の位相は 19 年で 1 周し，それを Meton 周期といい，その何年目かを示す．c は世紀．x は Gregorian 補正，西暦年数が 4 の倍数にもかかわらずうるう年にしなかった年の数である．z は太陰差補正で，1 朔望月を $29\frac{499}{940}$日 = 29.5308511 日で計算すると真の朔望月 = 29.530588 日より長く，2500 年に 8.13 日に積もる．その補正だ．e は歳首月齢 (epact)．ある年の歳首月齢を 0 とすると次の年の月齢は $356\frac{1}{4} - 12 \times 29\frac{499}{940} = 10\frac{827}{940}$だけ増える．この増分を繰り返し $29\frac{499}{940}$を超えたら $29\frac{499}{940}$を引く．n は 3 月 21 日の後の満月を見つける．春分満月は Paschal full moon という．

Gauss が自分の誕生日を知るために復活祭の式を考え出した話も有名だが，ここでは省略.

うるう月

46 巻 10 号の編集系独白で太陰太陽暦 (lunisolar calendar) のうるう月に触れた．これは各所に説明があり，「暦法算法の本」[1] では中国の暦の章にある．その規則は：

1. 新月の日 (朔の時刻を含む日) を決める．それをその月の 1 日とする.
2. その前日を前月の最後の日とする.
3. 太陽黄経が 0 度 (春分点) の時刻を含む月を 2 月，30 度を 3 月 , ..., 330 度を 1 月とする (**表 -2**).
4. 太陽黄経が 30 度の整数倍の時刻を含まぬ月をうるう月とする．直前が *m* 月なら閏 *m* 月とする .

表 -2 に 24 節気のうち太陽黄経が 30 度の整数倍になる中気といわれるものを示す.

節気	黄経	旧暦の月	節気	黄経	旧暦の月	節気	黄経	旧暦の月	節気	黄経	旧暦の月
雨水	330	1 月	小満	60	4 月	処暑	150	7 月	小雪	240	10 月
春分	0	2 月	夏至	90	5 月	秋分	180	8 月	冬至	270	11 月
穀雨	30	3 月	大暑	120	6 月	霜降	210	9 月	大寒	300	12 月

表-2　24節気のうち中気

これらは必ず旧暦の 1 月から 12 月に割り当てられ，うるう月にはならない．うるう月は Meton 周期の 19 年に 7 回置く．235 朔望月が 19 太陽年にほぼ等しい (29.53 日 × (12 × 19 + 7) = 6939.55 日, 365.2422 日 × 19 = 6939.60 日). 年表によると安政 1(1853) 年から明治 5(1872) 年の 19 年間にうるう月があったのは安政 1 年閏 7 月，安政 4 年閏 5 月，万延 1(1960) 年閏 3 月，文久 2(1862) 年閏 8 月，慶應 1(1865) 年閏 5 月，明治 1(1868) 年閏 4 月，明治 3 年閏 10 月の 7 回である.

国立天文台の暦要項に2006年の節気と朔望月のデータがある[5]ので借用する．以下のtimesの上4列は朔のGregorian暦による月日時分のリスト，下4列は節気の月日時分黄経のリストである（Haskellのリストは同じ型のものの並びなので，節気名の漢字は入れられない）．Listのモジュールをimportするとリストのソートができる．その結果も示す．一気にソートできるのが素晴らしい．

```
times = [[1,29,23,15],[2,28,9,31],[3,29,19,15],      --2006年3月29日は日食
         [4,28,4,44],[5,27,14,26],[6,26,1,5],
         [7,25,13,31],[8,24,4,10],[9,22,20,45],      --2006年9月22日は金環日食
         [10,22,14,14],[11,21,7,18],[12,20,23,1],
         [1,20,14,15,300],[2,19,4,26,330],[3,21,3,26,0],      -- 大寒， 雨水， 春分
         [4,20,14,26,30],[5,21,13,32,60],[6,21,21,26,90],     -- 穀雨， 小満， 夏至
         [7,23,8,18,120],[8,23,15,23,150],[9,23,13,3,180],    -- 大暑， 処暑， 秋分
         [10,23,22,26,210],[11,22,20,2,240],[12,22,9,22,270]]-- 霜降， 小雪， 冬至

Main> sort times
[[1,20,14,15,300],[1,29,23,15],[2,19,4,26,330],[2,28,9,31],[3,21,3,26,0],
 [3,29,19,15],[4,20,14,26,30],[4,28,4,44],[5,21,13,32,60],[5,27,14,26],
 [6,21,21,26,90],[6,26,1,5],[7,23,8,18,120],[7,25,13,31],[8,23,15,23,150],
 [8,24,4,10],[9,22,20,45],[9,23,13,3,180],[10,22,14,14],[10,23,22,26,210],
 [11,21,7,18],[11,22,20,2,240],[12,20,23,1],[12,22,9,22,270]]
```

これを見るとGreorian暦の1月29日から2月27日が旧暦の1月，春分をはさんで2月28日から3月28日が旧暦の2月，...ということが分かる．8月2日から9月21日までの間には節気がなく，これは閏7月だ．海上保安庁海洋情報部の新暦旧暦対照表と同じだ[6]．あたりまえか．ついでだが暦要項によると立春は2月4日8時27分で，これは旧暦の1月で，「年の内に春は来にけり」にはならない．しかし2006年はうるう月のため12月が遅くなり，2007年は年の内に春が来る．古今和歌集最初の有名な歌のおかげで，われわれは年の内に春が来るのは珍しいように感じているが実は存外多い．

太陽黄経の計算

毒を食らわばの感なきにしもあらずだが，ここまで来たので太陽黄経を計算してみる．月は難しいので割愛．天文年鑑にアルゴリズムが書いてあった．太陽について

平均黄経 $L = 280.4665 + 0.98564736 \times \mathrm{d} + 0.0003 \times T^2$

近地点黄経 $\tilde{\omega} = 282.9373 + 0.00004708 \times \mathrm{d} + 0.0005 \times T^2$

軌道離心率 $e = 0.016709 - 0.000042 \times T$

$d = \mathrm{JD} - 2451545.0$

$T = d/36525$ (Julian century)

平均近点離角 $M = L - \tilde{\omega}$

離心近点離角 E は e, M から Kepler 方程式 $M = E - e \sin E$ を解いて得る．

真近点離角 V は $\tan(V/2) = \sqrt{\frac{1+e}{1-e}}\ \tan(E/2)$

黄経引数 $U = \tilde{\omega} + V$

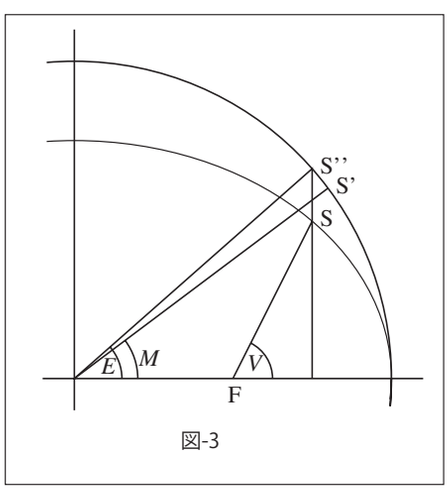

図-3

$L, \tilde{\omega}, M, E, V, U$ の単位は度．JDは黄経を計算したい時刻のJulian Date．**図 -3** で楕円は黄道；地球，太陽はそれぞれF, Sにある．

平均黄経 L は太陽 (S') が円軌道を等速で回っているとした角度，$0.98564736 \times d$ から分かるように1日に1度弱

移動する (360 / 365.2422 => 0.98564733). 第 1 項の 280.4665 は d で 2000 年 1 月 1 日の JD を引いていることからその基準日の太陽黄経. $\tilde{\omega}$ は近日点の黄経. その L との差が M. これを面積速度一定の楕円軌道に修正をする (S' → S''). その角度が E で M から Newton 法で解く. E の初期値を適当に決め, $E' = E - \frac{E - e\sin E - M}{1 - e\cos E}$ を繰り返す. ここだけは弧度法で計算する. E が決まれば V が得られる .

```
sind d = sin (d * pi / 180)          -- 度単位の三角関数
cosd d = cos (d * pi / 180)
tand d = tan (d * pi / 180)
datan x = 180 / pi * atan x

ld d = norm (280.4665 + 0.98564736 * d + 0.0003 * t * t) 360-- 平均黄経 l(d)
       where t = d / 36525
omegad d = 282.9373 + 0.00004708 * d + 0.0005 * t * t        -- 近地点黄経 omage(d)
       where t = d / 36525
ed d = 0.016709 - 0.000042 * t       -- 離心率 e(d)
       where t = d / 36525

norm :: Float -> Integer -> Float    -- 0 〜 360 度に正規化
norm a d = fromInteger (b `mod` d) + c
   where (b,c) = properFraction a    -- b,c は a の整数部と小数部
lon td =                             -- td(JD) における太陽黄経を計算
  norm l 360   --td = julian date of the day time
   where d = td - 2451545.0          -- 2000 年 1 月 1 日正午 (UT) からの日数
         e = ed d                    -- その時点での離心率
         omega = omegad d            -- その時点での近地点黄経
         m = ld d - omega            -- 平均近点離角
         e' = kepler e m             -- 離心近点離角
         kepler e m =                -- Kepler 方程式を解く
           f e0
            where m' = m * pi / 180 -- 弧度法に直す
                  e0 = m'            -- Newton 法の初期値
                  f e0 =
                   if abs (e0 - e1) < 0.00001 then (e1 * 180 / pi) else (f e1)
                     where e1 = e0 - (e0 - e * sin e0 - m')/(1 - e * cos e0)
         l = v e m + omega           -- 太陽黄経
           where v e m = 2 * datan (sqrt ((1 + e) / (1 - e)) * tand (e'/2))
```

しかしこの種の計算の精度を上げるのはきわめて難しい. 2004 年 1 月 0 日 0 時正子 UT の太陽黄経は

```
Main> lon (fromIntegral (julianDate 2004 1 0) - 0.5) -- JD は正午が基準ゆえ 0.5 を引く
278.8828
```

と得られる. 一方, 天文年鑑や理科年表では 278.825 である. 「暦法算法の本」には章動 (nutation) や光行差 (aberration) を考慮に入れた太陽黄経の式があり, それも試みたがそれでも理科年表の値は得られなかった. 素人の手には余るものかもしれない.

参考文献

1) Reingold, E. M. and Dershowitz, N. : Calendrical Calculations : The Millenium Edition, Cambridge University Press (2001).
2) 青木信仰: 時と暦, 東京大学出版会 (1982).
3) 島内剛一: 万年七曜表, 数学セミナー , 1978 年 7 月号, pp.35-48 (1978).
4) Bird, R. : Introduction to Functional Programming Using Haskell, 2nd ed, Prentice Hall (1998).
5) 暦要項, http://www.nao.ac.jp/koyomi/yoko/
6) 新暦旧暦対照表, http://www1.kaiho.mlit.go.jp/KOHO/reki/kyuu9700.htm

(平成 17 年 11 月 1 日受付)