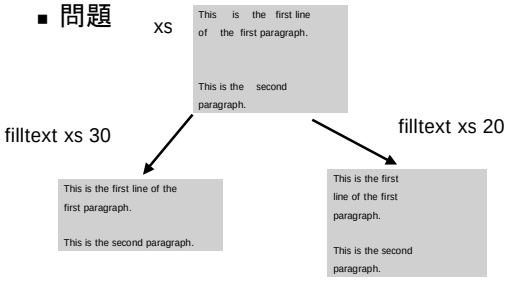
リスト処理の例（その2）

胡　振江

例題3：テクスト処理

- 問題



行の列としてのテキスト

```
type Line' = [Char]

隣接する行と行の間に改行文字を挿入し、それらを連接する
unlines' = foldr1 oplus
    where xs `oplus` ys = xs ++ "¥n" ++ ys

文字のリストと考えられるテクストを行の列に変換する
lines' = foldr otimes [[]]
    where x `otimes` xss
                    | x=='¥n'   = [[]] ++ xss
                    | otherwise = [[x] ++ head xss] ++ tail xss
```

語の列としての行

```
type Word' = [Char]

隣接する語と語の間に空白文字を挿入し、それらを連接する
unwords' = foldr1 oplus
    where xs `oplus` ys = xs ++ " " ++ ys

行を語に分割する
words' = filter (/=[]) . foldr otimes [[]]
    where x `otimes` xss
                    | x==' '   = [[]] ++ xss
                    | otherwise = [[x] ++ head xss] ++ tail xss
```

行の列と段落

```
type Para = [Line']

隣接する段落と段落の間に空の行を挿入し、それらを連接する
unparas = foldr1 oplus
    where xs `oplus` ys = xs ++ [[]] ++ ys

行の列を分割して段落の列にする
paras = filter (/=[]) . foldr otimes [[]]
    where xs `otimes` xss
                    | xs==[]    = [[]] ++ xss
                    | otherwise = [[xs] ++ head xss] ++ tail xss
```

基本的なテキスト処理関数

```
countlines = length . lines'
countwords = length . concat . map words' . lines'
countparas = length . paras . lines'

normalise :: Text' -> Text'
normalise = unparse . parse

parse :: Text' -> [[[Word']]]
parse = map (map words') . paras . lines'

unparse :: [[[Word']]] -> Text'
unparse = unlines' . unparas . map (map unwords')
```

応用:段落の詰め込み

```
filltext m = unparse . map (fill m) . testparas
testparas = map linewords . paras  . lines'
linewords = concat . map words‘

1行に収まるように最長の語の列を取る
fill m [] = []
fill m ws = [fstline] ++ fill m restwds
    where fstline = take n ws
          restwds = drop n ws
          n = greedy m ws

greedy m ws = maximum [ length us | us <- inits ws,
                                length (unwords' us) <= m ]
```

例題4:カレンダーの印刷

- 問題:calendar 2002 ➔

```
JANUARY 2002          FEBRUARY 2002         MARCH 2002

Sun    6 13 20 27     Sun    3 10 17 24     Sun    3 10 17 24 31
Mon    7 14 21 28     Mon    4 11 18 25     Mon    4 11 18 25
Tue  1  8 15 22 29    Tue    5 12 19 26     Tue    5 12 19 26
Wed  2  9 16 23 30    Wed    6 13 20 27     Wed    6 13 20 27
Thu  3 10 17 24 31    Thu    7 14 21 28     Thu    7 14 21 28
Fri    4 11 18 25     Fri    1  8 15 22     Fri    1  8 15 22 29
Sat    5 12 19 26     Sat    2  9 16 23     Sat    2  9 16 23 30

APRIL 2002            MAY 2002              JUNE 2002

Sun    7 14 21 28     Sun    5 12 19 26     Sun    2  9 16 23 30
Mon  1  8 15 22 29    Mon    6 13 20 27     Mon    3 10 17 24
.....
```

抽象的なカレンダーの構成

↓

カレンダーの印刷

図形の表示

```
type Picture = [[Char]]

height,width :: Picture -> Int
height p = length p
width p = length (head p)
```

```
1 2 3 4
5 6 7 8
```

↓

```
[[‘1’,‘2’,‘3’,‘4’],
 [‘5’,‘6’,‘7’,‘8’]]
```

図形の構成

```
図形qの上に図形pを置く
p `above` q | width p == width q = p++q
図形pを図形qの左に置く
p `beside` q | height p == height q = zipWith (++) p q

図形のリストを縦に積む
stack = foldr1 above
図形リストを横に並べる
spread = foldr1 beside

特定の高さと幅をもつ空の図形の生成
empty (h,w) = copy (copy ' ' w) h
```

図形のgrouping

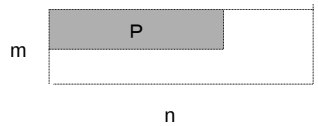
```
block :: Int -> [Picture] -> Picture
block n = stack . map spread . group n
group n xs = [take n (drop j xs) | j <- [0,n..(length xs-n)]]

[G1,G2,G3,G4,G5,G6,G7,G8] ➔   G12
                        n=2   G34
                              G56
                              G78

blockT :: Int -> [Picture] -> Picture
blockT n = spread . map stack . group n
```

図形の埋め込み

```
高さm,幅nの大きな図形の左上部に図形pをはめ込む
lframe (m,n) p = (p `beside` empty (h,n-w))
                    `above` empty (m-h,n)
   where h = height p
         w = width p
```



カレンダーの表示

```
picture (mn,yr,fd,ml) = title mn yr `above` table fd ml

各月の見出し
title mn yr = lframe (2,25) [mn ++ " " ++ show yr]

table fd ml = lframe (8,25) (daynames `beside` entries fd ml)
daynames = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

entries fd ml = blockT 7 (dates fd ml)
dates fd ml = map (date ml) [(1-fd)..(42-fd)]
date ml d | d<1 || ml < d = [rjustify 3 " "]
          | otherwise     = [rjustify 3 (show d)]
```

カレンダーの作成

```
calendar :: Int -> String
calendar = display . block 3 . map picture . months

months yr = zip4 mnames (copy yr 12) (fstdays yr)
                 (mlengths yr)
  where zip4 [] [] [] [] = []
        zip4 (x:xs) (y:ys) (z:zs) (u:us)
             = (x,y,z,u) : zip4 xs ys zs us

display = unline
```

例題4:亀の子幾何

- 問題

```
[down, move, move,
 right, move, move,
 up, move, move,down]
```

```
*** *
*
*
```

亀の子の状態の表現

```
type State = (Direction,Pen,Point)

type Direction = Int  -- 0:North, 1:East, 2:South | West

type Pen = Bool

type Point = (Int,Int)
```

コマンド

```
type Command = State -> State
move :: Command
move (0,p,(x,y)) = (0,p,(x-1,y))    -- W -----y-----> E
move (1,p,(x,y)) = (1,p,(x,y+1))    -- N -----x-----> S
move (2,p,(x,y)) = (2,p,(x+1,y))
move (3,p,(x,y)) = (3,p,(x,y-1))
right,left :: Command
right (d,p,(x,y)) = ((d+1) `mod` 4,p,(x,y))
left (d,p,(x,y)) = ((d-1) `mod` 4,p,(x,y))
up,down :: Command
up (d,p,(x,y)) =  (d,False,(x,y))
down (d,p,(x,y)) = (d,True,(x,y))
```

状態列の生成

```
turtle :: [Command] -> [State]
turtle = scanl applyto (0,False,(0,0))
  where applyto x f = f x
```

状態列の表示

```
display :: [Command] -> [Char]
display = layout . picture . trail . turtle

trail :: [State] -> [Point]
trail ss = [(x,y) | (_,p,(x,y)) <- ss, p]

picture :: [Point] -> [[Char]]
picture = symbolise . bitmap
    where symbolise = map (map mark)
          mark True = '*'
          mark False = ' '
```

ビットマップの生成

```
bitmap ps = [[(x,y) `elem` ps | y<-yran] | x<-xran]
    where xran = range' (map fst ps)
          yran = range' (map snd ps)
range' xs = range (minimum xs, maximum xs)
```