リスト処理の例

胡　振江

例題1：数をことばに

- 問題：
  0以上100万以下の数 ➔ 通常の英語表現
  例：
  - 308000 ➔ three hundred and eight thousand
  - 369027 ➔ three hundred and sixty-nine thousand and twenty-seven
  - 369401 ➔ three hundred and sixty-nine thousand four hundred and one

解決法

- 簡単な問題から複雑問題へ
  - n<100 の数字を対象に
  - n<1000 の数字を対象に
  - n< 1000,000 の数字を対象に

数の英語名：文字列

```
units = [ "one", "two", "three", "four", "five",
          "six", "seven", "eight", "nine"]

teens = ["ten", "eleven", "twelve", "thirteen",
         "fourteen", "fifteen", "sixteen",
       "seventeen", "eighteen", "nineteen"]

tens = ["twenty", "thirty", "forty", "fifty", "sixty",
        "seventy", "eighty", "ninety"]
```

0<n<100の場合

```
convert2 n = combine2 (digits2 n)

digits2 n = (n `div` 10, n `mod` 10)

combine2 (0,u+1)     = units !! u
combine2 (1,u)       = teens !! u
combine2 (t+2,0)     = tens !! t
combine2 (t+2,u+1)   = tens !! t ++ "-" ++
                       units !! u
```

0<n<1000の場合

```
convert3 n = combine3 (digits3 n)

digits3 n = (n `div` 100, n `mod` 100)

combine3 (0,t+1) = convert2 (t+1)
combine3 (h+1,0) = units !! h ++ " hundred"
combine3 (h+1,t+1) = units !! h ++ " hundred
                     and " ++ convert2 (t+1)
```

0<n<1000,000の場合

```
convert6 n = combine6 (digits6 n)
digits6 n = (n `div` 1000, n `mod` 1000)

combine6 (0,h+1) = convert3 (h+1)
combine6 (m+1,0) = convert3 (m+1) ++ " thousand"
combine6 (m+1,h+1) = convert3 (m+1) ++
                     " thousand“ ++
                     link (h+1) ++
                      convert3 (h+1)

link h | h < 100 = " and "
       | otherwise = " "
```

実行例

```
Convert> convert6 308000
"three hundred and eight thousand"
(985 reductions, 1350 cells)

Convert> convert6 369027
"three hundred and sixty-nine thousand and twenty-seven"
(1837 reductions, 2547 cells)

Convert> convert6 369401
"three hundred and sixty-nine thousand four hundred and one"
(1851 reductions, 2548 cells)
```

例題2：可変長の算術演算

- 問題：
  任意の大きさの整数計算を行う関数パッケージを作る。
  - 比較：[2,1,3,4] > [3]
  - 加算：[7,3,7] + [4,6,9] = [1,2,0,6]
  - 減算：[4,0,6] – [3,7,5] = [3,1]
  - 乗算：[1,2] * [1,5] = [1,8,0]
  - 除算：[1,7,8,4] ÷ [6,2] = [2,8] ... [4,8]

可変長整数の表現 (1)

- リストでの表現:

$$[x_{n-1}, x_{n-2}, \ldots, x_0] = \sum_{k=0}^{n-1} x_k b^k$$

例：b = 10000の場合：

123456789 => [1,2345,6789]

100020003 => [1,2,3]

```
type VInt = [Bigit]
type Bigit = Int
```

可変長整数の表現 (2)

- 標準形
  strep: 必要でない0を取り除く.

strep [0,0,1,2] = [1,2]

```
strep xs | ys == []  = [0]
         | otherwise = ys
  where ys = dropWhile (==0) xs
```

比較演算 (1)

比較を行う前に2つの数の桁数を合わせる.

```
align xs ys | n>0 = (copy 0 n ++ xs, ys)
            | otherwise = (xs, copy 0 (-n) ++ ys)
  where  n = length ys - length xs

copy x n = [ x | j <- [1..n] ]
```

比較演算 (2)

```
vcompare :: (VInt->VInt->Bool) -> VInt -> VInt -> Bool
vcompare op xs ys = op us vs
    where (us,vs) = align xs ys

veq = vcompare (==)
vleq = vcompare (<=)
vless = vcompare (<)
```

加算と減算 (1)

- 加算

b-数ごとに加え合わせる　正規化する

例: [7,3,7] + [4,6,9] => [11,9,16] => [1,2,0,6]

正規化: それぞれのb-数の桁をb法として縮小し，手前の桁から繰上げていく.

[x1, x2, …, xn] => carry x1 (carry x2 … (carry xn [0]))
(= foldr carry [0] [x1,x2,…,xn])

```
norm = strep . foldr carry [0]
  where carry :: Bigit -> VInt -> VInt
        carry x (c:xs) = (x+c) `div` b : (x+c) `mod` b : xs
```

加算と減算 (2)

- 加算

b-数ごとに加え合わせる　正規化する

例: [7,3,7] + [4,6,9] => [11,9,16] => [1,2,0,6]

```
vadd xs ys = norm (zipWith (+) xs' ys')
    where (xs',ys') = align xs ys
```

加算と減算 (3)

- 減算

b-数ごとに引く　正規化する

例: [1,0,6] + [3,7,5] => [-2,-7,1] => [-1,7,3,1]

```
vsub xs ys = norm (zipWith (-) xs' ys')
    where (xs',ys') = align xs ys
```

符号反転する関数

符号の判定:

```
negative xs = head xs < 0
```

符号の反転:

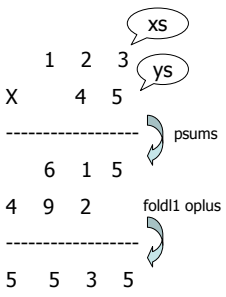
```
vnegate = norm . map neg
neg x = -x
```

例: vnegate [-1,7,3,1] = [2,6,9]

乗算

```
vmul xs ys = foldl1 oplus (psums xs ys)
  where
    psums xs ys =
      [norm (map (y*) xs) | y<-ys]
    xs `oplus` ys = vadd (xs++[0]) ys
```

例: vmul [1,2,3] [4,5] = [5,5,3,5]



除算：商と余り (1)

商と余りを求めるアルゴリズムは
　商の1桁を求められ、
　次の桁のための余りが計算される
という計算段階を繰り返して行うものである

その結果：[(q0,rs0),(q1,rs1),…,(qn,rsn)]
- 商：　[q0,q1,…qn]
- 余り：rsn

除算：商と余り (2)

例：[1,7,8,4] / [6,2]
(0, [1])
⇩ (dstep [6,2]) r 7
(0, [1,7])
⇩ (dstep [6,2]) r 8
(2, [5,4])
⇩ (dstep [6,2]) r 4
(8, [4,8])

```
divalg xs ys
  = scanl
      (dstep ys)
      (0,take m xs)
      (drop m xs)
```

商：[0,0,2,8]　余：[4,8]

dstep

dstepの定義：
- 被除数xsの長さが除数ysの長さより短いか
- または、等しいか
- または、それより長いか

```
dstep ys (q,rs) x
                | length xs < length ys        = astep xs ys
                | length xs == length ys       = bstep xs ys
                | length xs == length ys + 1 = cstep xs ys
        where xs = rs ++ [x]
```

astep, bstepの定義

1 被除数xsの長さが除数ysの長さより短い

```
astep xs ys = (0,xs)
```

2 被除数xsの長さが除数の長さと等しい

```
bstep xs ys | negative zs = (0,xs)
            | otherwise   = (1,zs)
  where zs = vsub xs ys
```

条件: head ys >= b `div` 2

cstepの定義

3. 被除数xsの長さが除数ysの長さより長い
　　q'-2 <= q <= q'
　ここで、q' = min ((x0*b+x1) `div` y1) (b-1)

条件：y1 >= b / 2

```
cstep xs ys | vless rs0 ys = (q,rs0)
            | vless rs1 ys = (q+1,rs1)
            | otherwise    = (q+2,rs2)
      where rs0 = vsub xs (bmul ys q)
            rs1 = vsub rs0 ys
            rs2 = vsub rs1 ys
            q = guess xs ys - 2
```

条件「y1>=b/2」を満たすために

除数ysの先頭のb-数y1を十分大きくするために，
適当な尺度因子dを除数と被除数にかける．

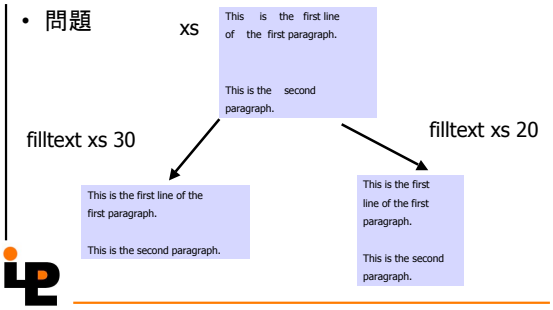
```
vqrm [x1] [y1] = (x1 `div` y1, y1 - y1 * (x1 `div` y1)]

vqrm xs ys = (strep qs, strep rs)
      where qs = map fst ds
            rs = bdiv (snd (last ds)) d
            ds = divalg (bmul xs d) (bmul ys d)
            d  = b `div` (head ys + 1)

bdiv xs d = vqrm xs [d]
```

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

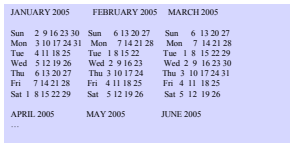
```
filltext m = unparse . map (fill m) . testparas
testparas = map linewords . paras  . lines'
linewords = concat . map words'

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 2005 ➔



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

図形:同じ長さの文字列のリストを要素に持つリストで表現される.

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:図示演算子の定義

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関数

```
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] ➔ n=2

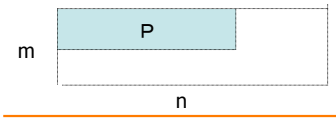
G1 G2
G3 G4
G5 G6
G7 8G

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

図形のはめ込み

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
```



図形の印刷

```
display :: Picture -> String
display = foldr1 (insert '¥n')
  where insert n x r = x ++ [n] ++ r
```

例: display ["123", "456", "789"]
=> "123¥n456¥n789"

カレンダーの図示

```
month_pic (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 month_pic . 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
```

カレンダーの印刷

```
> putStrLn (calender 2004)
```

レポートの提出について

- 課題内容:
  - 演習問題 4.5.1, 4.5.2, 4.5.3, 4.5.4, 4.5.5, 4.5.6, 4.5.7を解け.
- 注意事項:
  - 実行できるソース・実行例を添付すること.
  - 締切日:1月 10日(火)
  - 提出先:胡のポストへ

報告書に名前と学生証番号を忘れずに記入すること