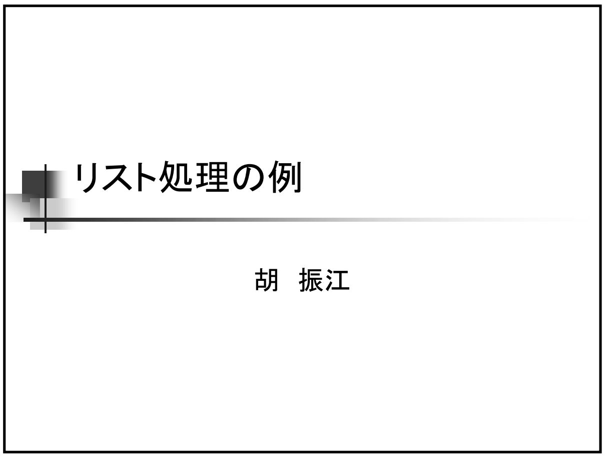
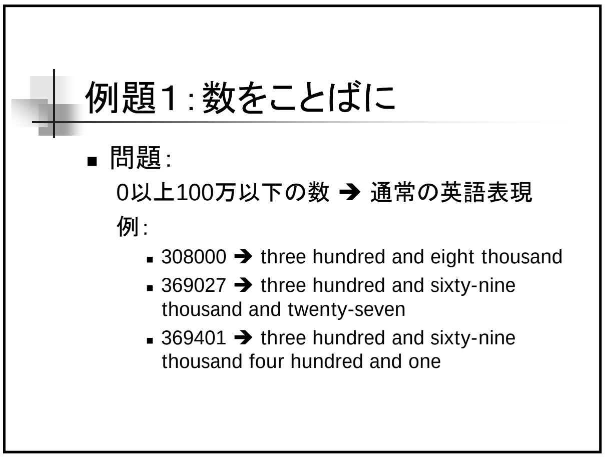


リスト処理の例

胡　振江

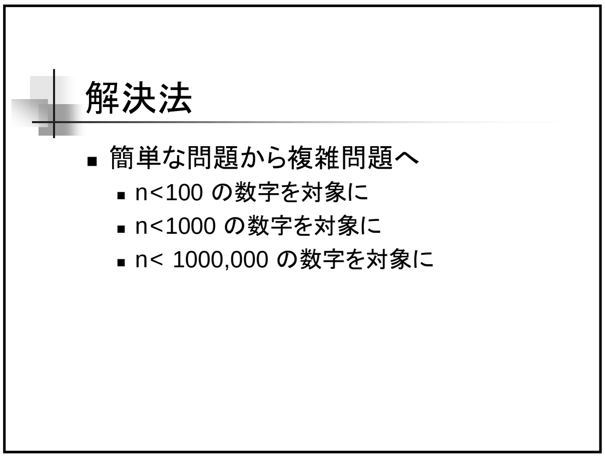


例題1：数をことばに

- 問題：

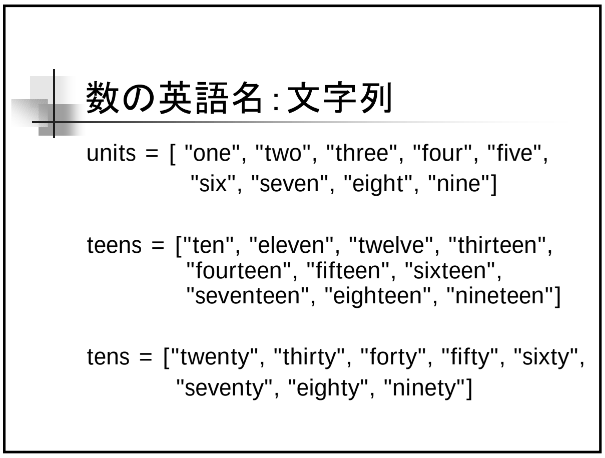
  0以上100万以下の数 ➔ 通常の英語表現

  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



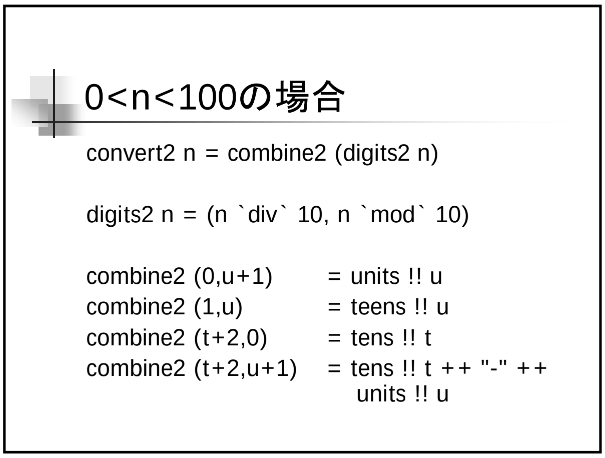
解決法

- 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



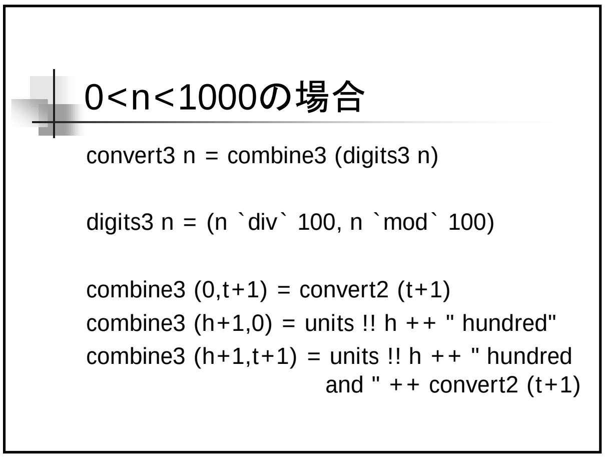
数の英語名：文字列

```
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"]
```



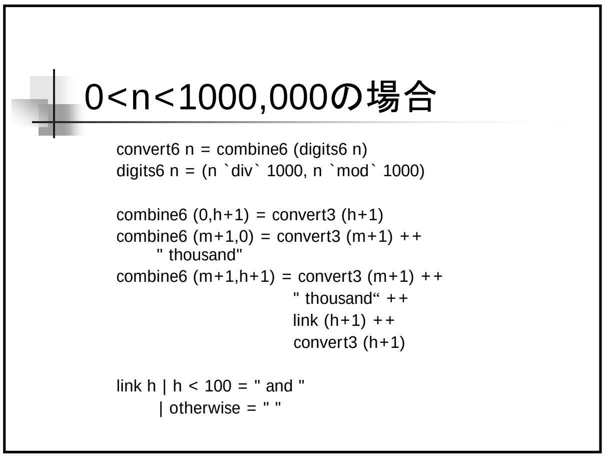
0<n<100の場合

```
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
```



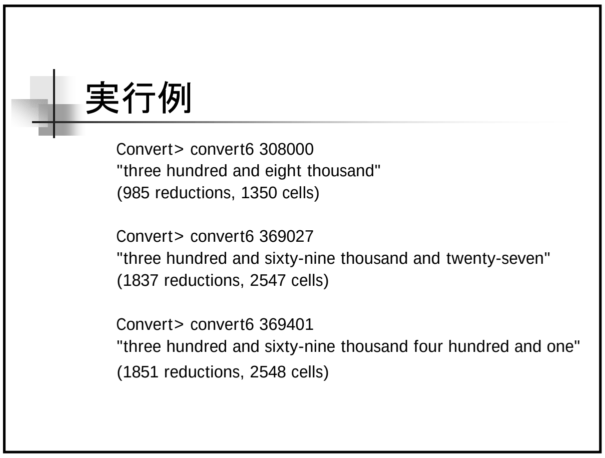
0<n<1000の場合

```
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)
```



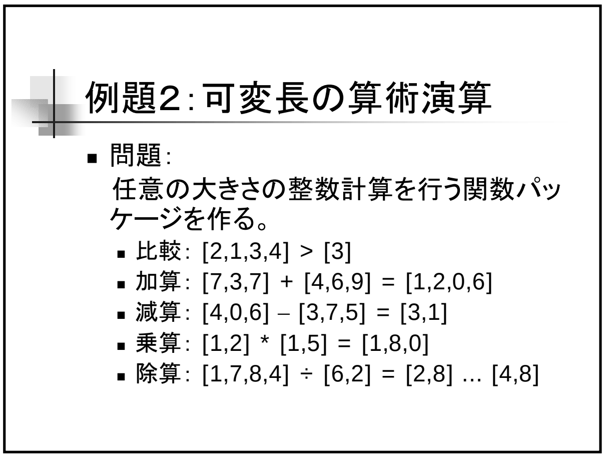
0<n<1000,000の場合

```
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 = " "
```



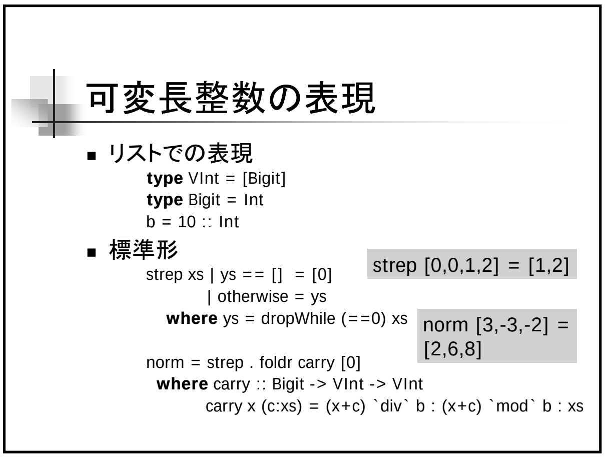
実行例

```
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

- リストでの表現

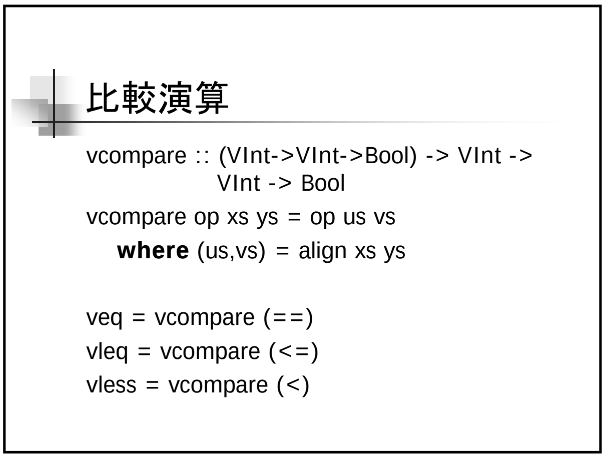
```
type VInt = [Bigit]
type Bigit = Int
b = 10 :: Int
```

- 標準形

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

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

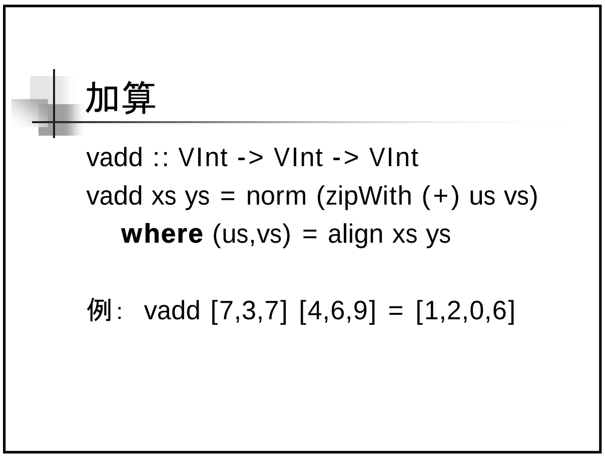
strep [0,0,1,2] = [1,2]

norm [3,-3,-2] = [2,6,8]



比較演算

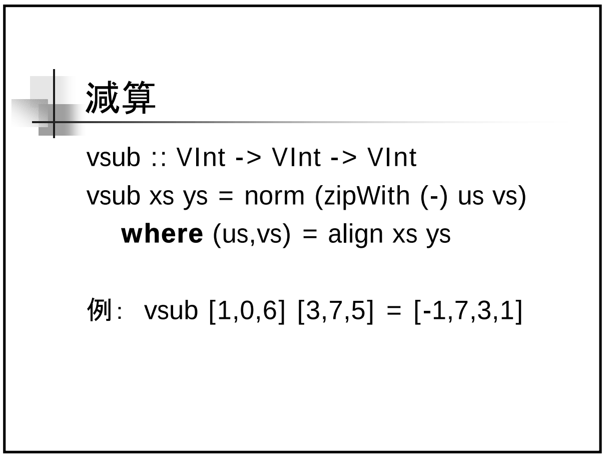
```
vcompare :: (VInt->VInt->Bool) -> VInt ->
            VInt -> Bool
vcompare op xs ys = op us vs
   where (us,vs) = align xs ys

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



加算

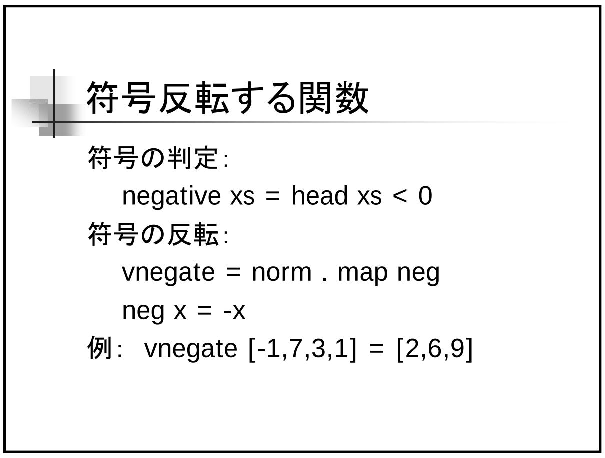
```
vadd :: VInt -> VInt -> VInt
vadd xs ys = norm (zipWith (+) us vs)
   where (us,vs) = align xs ys
```

例： vadd [7,3,7] [4,6,9] = [1,2,0,6]



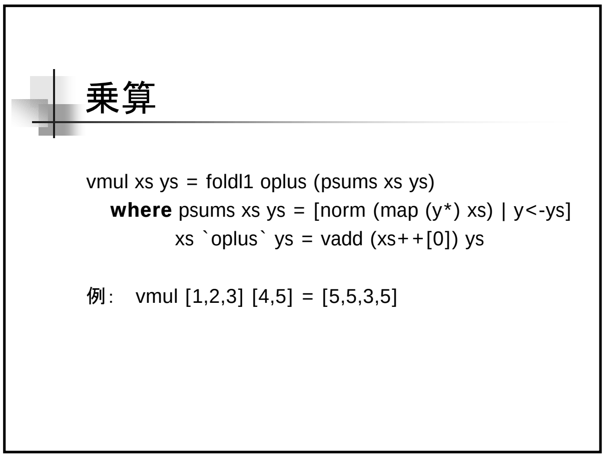
減算

```
vsub :: VInt -> VInt -> VInt
vsub xs ys = norm (zipWith (-) us vs)
   where (us,vs) = align xs ys

例:  vsub [1,0,6] [3,7,5] = [-1,7,3,1]
```



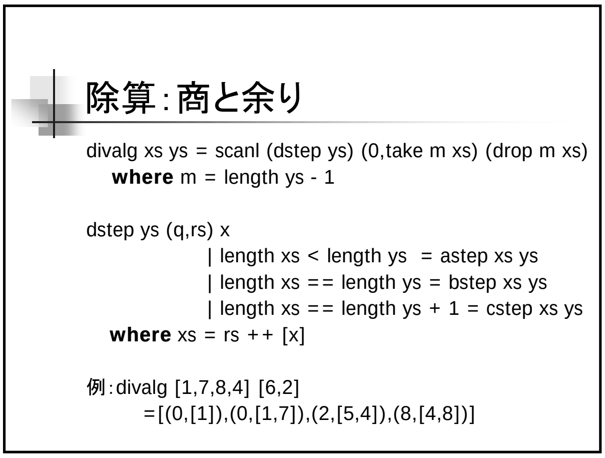
符号反転する関数

```
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]
```



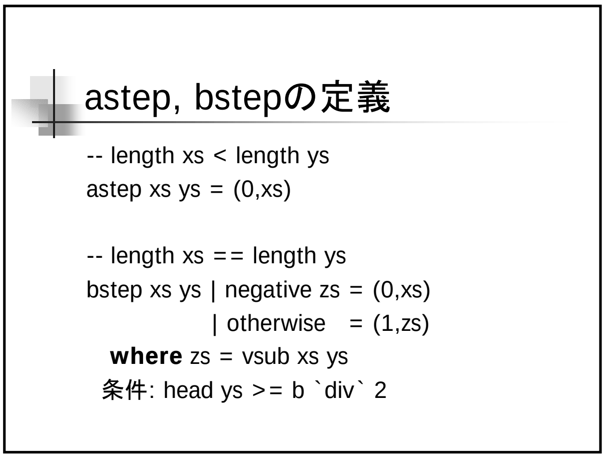
乗算

```
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]
```



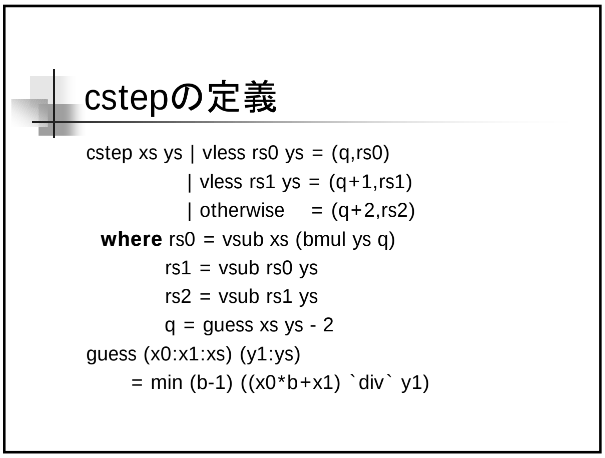
除算:商と余り

```
divalg xs ys = scanl (dstep ys) (0,take m xs) (drop m xs)
    where m = length ys - 1

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]

例:divalg [1,7,8,4] [6,2]
       =[(0,[1]),(0,[1,7]),(2,[5,4]),(8,[4,8])]
```



astep, bstepの定義

```
-- length xs < length ys
astep xs ys = (0,xs)

-- length xs == length ys
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

```
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
guess (x0:x1:xs) (y1:ys)
      = min (b-1) ((x0*b+x1) `div` y1)
```

条件を満たすように

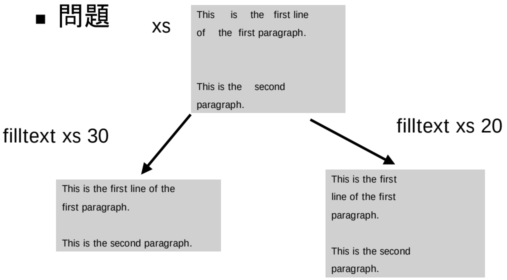
```
vqrm xs ys = (strep qs, strep rs)
  where qs = map fst ds
        rs = bdiv (snd (last ds)) d
        ds = divalg (bmul xs d) (bmul ys d)
        d  = b `div` (head ys + 1)
```

bdiv: 数を1桁のb数で割る関数

```
bqrm [x] d = ([x `div` d], x `mod` d)
bqrm (x:xs) d = (strep qs, last rs `mod` d)
  where qs = map (`div` d) rs
        rs = scanl oplus x xs
        r `oplus` x = b * (r `mod` d) + x

bdiv xs d = fst (bqrm xs d)
```

例題3：テクスト処理

- 問題



xs

This is the first line
of the first paragraph.

This is the second
paragraph.

filltext xs 30

This is the first line of the
first paragraph.

This is the second paragraph.

filltext xs 20

This is the first
line of the first
paragraph.

This is the second
paragraph.

行の列としてのテキスト

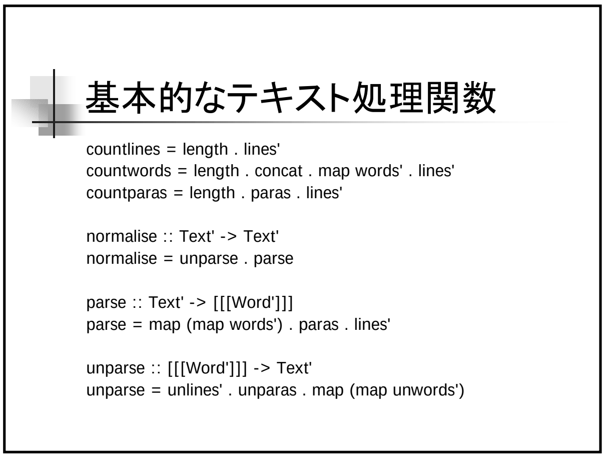
```
type Line' = [Char]
lines' :: Text' -> [Line']
unlines' :: [Line'] -> Text'

unlines' = foldr1 oplus
  where xs `oplus` ys = xs ++ "¥n" ++ ys

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]
words' :: Line' -> [Word']
unwords' :: [Word'] -> Line'

unwords' = foldr1 oplus
  where xs `oplus` ys = xs ++ " " ++ ys

words' = filter (/=[]) . foldr otimes [[]]
  where x `otimes` xss
                    | x==' '    = [[]] ++ xss
                    | otherwise = [[x] ++ head xss] ++ tail xss
```

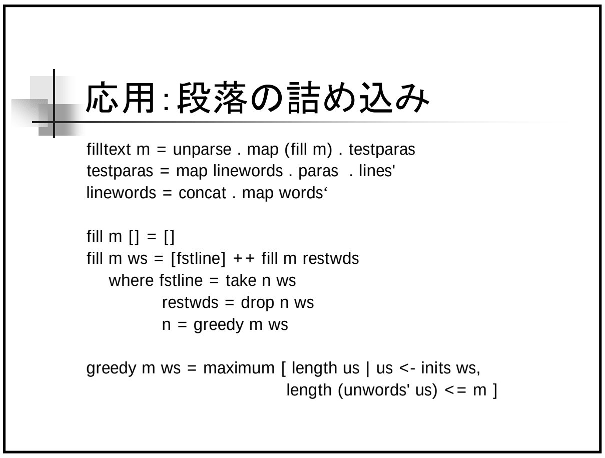
行の列と段落

```
type Para = [Line']
paras :: [Line'] -> [Para]
unparas :: [Para] -> [Line']

unparas = foldr1 oplus
  where xs `oplus` ys = xs ++ [[]] ++ ys

paras = filter (/=[]) . foldr otimes [[]]
  where xs `otimes` xss
                    | xs==[]    = [[]] ++ xss
                    | otherwise = [[xs] ++ head xss] ++ tail xss
```



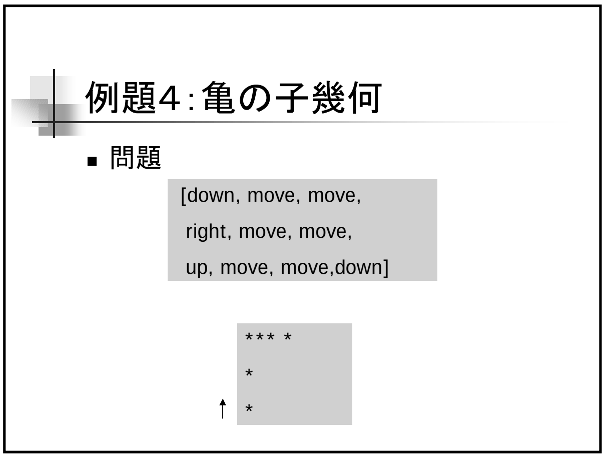
基本的なテキスト処理関数

```
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‘

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 ]
```

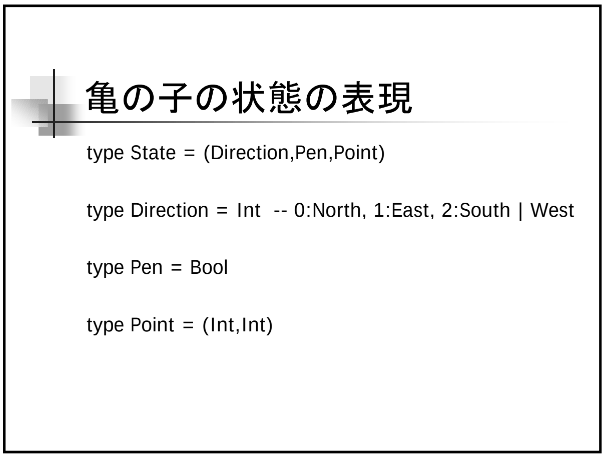


例題4：亀の子幾何

- 問題

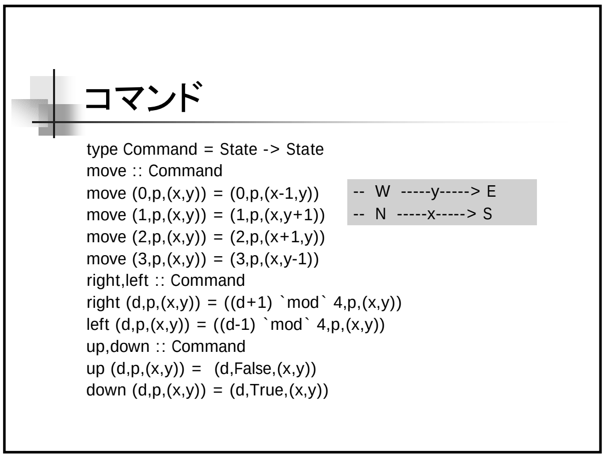
```
[down, move, move,
 right, move, move,
 up, move, move,down]
```

```
  *** *
  *
↑ *
```



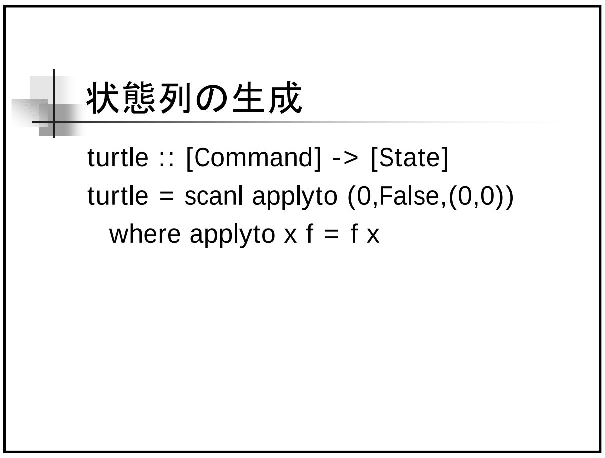
亀の子の状態の表現

```
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)
```

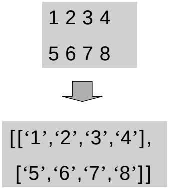
例題5：カレンダーの印刷

- 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
.......
```

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
```

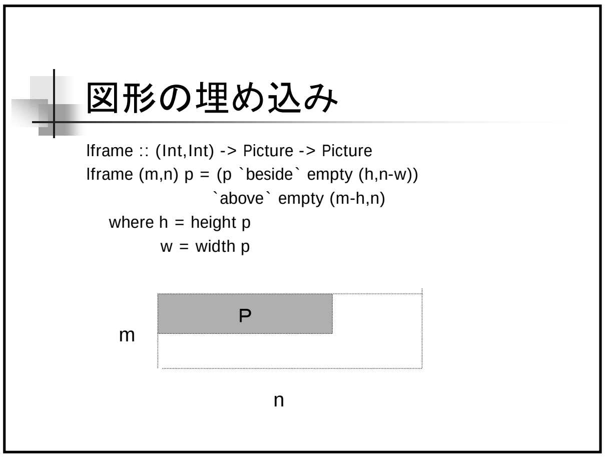
```
[['1','2','3','4'],
 ['5','6','7','8']]
```

図形の構成

```
above,beside :: Picture -> Picture -> Picture
p `above` q | width p == width q = p++q
p `beside` q | height p == height q = zipWith (++) p q

stack,spread :: [Picture] -> Picture
stack = foldr1 above
spread = foldr1 beside

empty :: (Int,Int) -> Picture
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
```



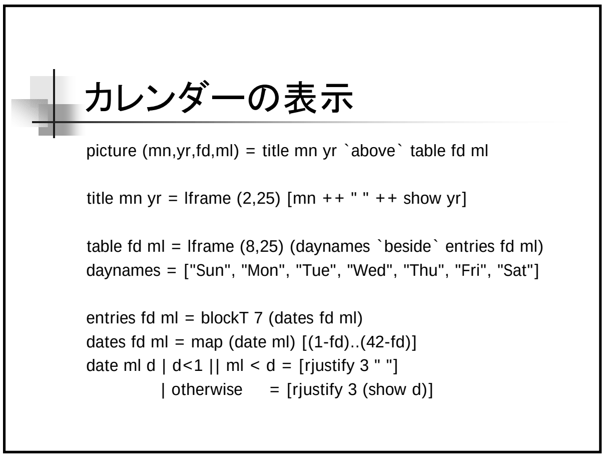
図形の埋め込み

```
lframe :: (Int,Int) -> Picture -> Picture
lframe (m,n) p = (p `beside` empty (h,n-w))
                 `above` empty (m-h,n)
   where h = height p
         w = width p
```

P

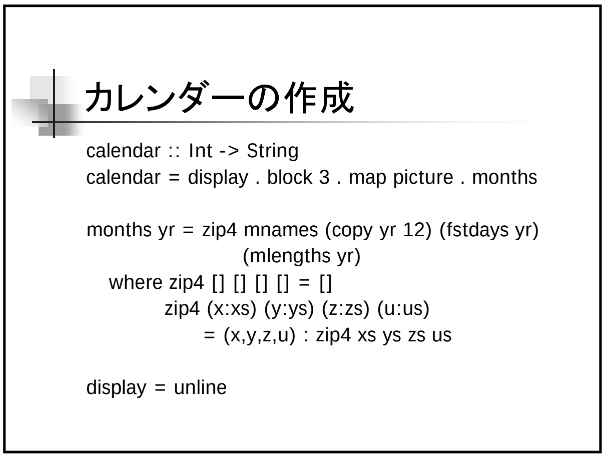
m

n



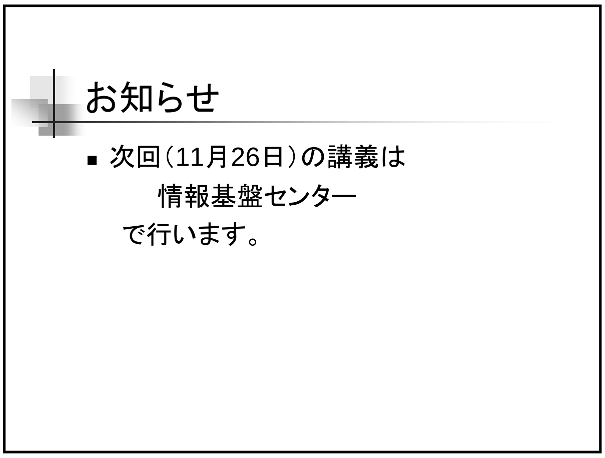
カレンダーの表示

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

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
```



お知らせ

- 次回（11月26日）の講義は
  情報基盤センター
  で行います。