リスト処理の例（その１）

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

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

divalg

- 被除数 xs = [x1,x2,…,xn]
- 除数 ys = [y1,y2,…,ym]

  - (q0,rs0) = (0,[x1,…,x(m-1)])
  - (qk,rsk), ys, x(k+m-1) ➔ q(k+1),rs(k+1)

    dstep

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

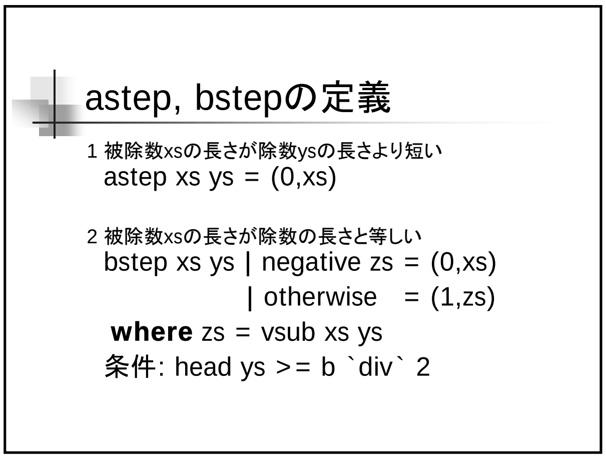
例:divalg [1,7,8,4] [6,2]
=[(0,[1]),(0,[1,7]),(2,[5,4]),(8,[4,8])]

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

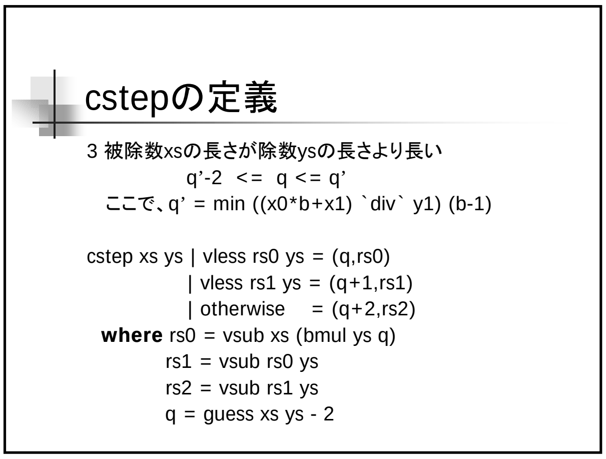
```
dstep ys (q,rs) x
            | length xs < length ys  = astep xs ys
            | length xs == length ys = bstep xs ys
            | length xs == length ys + 1 = cstep xs ys
  where xs = rs ++ [x]
```



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)
```

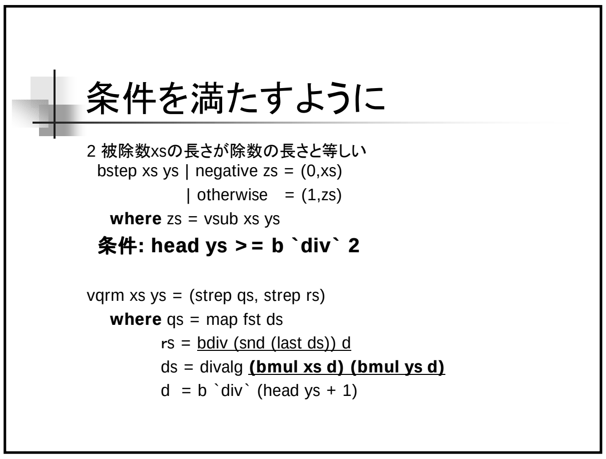
2 被除数xsの長さが除数の長さと等しい
```
bstep xs ys | negative zs = (0,xs)
            | otherwise   = (1,zs)
 where zs = vsub xs ys
条件: head ys >= b `div` 2
```



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)

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
```



条件を満たすように

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

```
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)
```