課題 5

与えられた数を、日本語の金額で表す関数 convert を定義しなさい。

解答及び解説

まずは、単純に数字を漢数字に変換するプログラムを見ることにする。漢数字は位取り記数法ではないので、位取りのための 0 がない。従って、関数 convert では、0 だけを特別扱いとした。

```
convert :: Int -> IO ()
convert 0 = putStrLn "金零円"
convert n = putStrLn ("金" ++ conv1 n classes ++ "円")
```

さて、漢数字を使うと 4 桁ごとに区切った数に「万」「億」を付すことになる。関数 conv1 がそれを行う。次のように定義する。

```
classes = [ "", "万", "億"]

conv1 :: Int -> [String] -> String
conv1 0 cls = ""
conv1 n (cl:cls) | m == 0    = conv1 d cls
                 | otherwise = conv1 d cls ++ conv2 m scales ++ cl
        where d = div n 10000
              m = mod n 10000
```

where 節では、4 桁ごとに区切るため、与えられた数 n を 10000 で割ったときの商と余りを求めている。これは関数 divMod を使うと、

```
where (d, m) = divMod n 10000
```

と書くことができる。

次に、conv1 で利用されている関数 conv2 は、4 桁の数を漢数字で表現する。

```
digits = ["", "壱", "弐", "参", "四", "五", "六", "七", "八", "九"]
scales = ["", "拾", "百", "千"]

conv2 :: Int -> [String] -> String
conv2 0 ss = ""
conv2 n (s:ss) | m == 0    = conv2 d ss
               | otherwise = conv2 d ss ++ (digits!!m) ++ s
        where (d, m) = divMod n 10
```

以上で、とりあえず動作するものが出来た。

ところで、注意書きに「10 億円まで」とある理由は、2 の 32 乗つまり標準的なコンピュータの扱う整数の範囲 (おおよそ 20 億程度) を考慮してのことである。多くのプログラミング言語では、標準の整数の型がこの範囲である。

そこで、上記のプログラムでは、整数の型として Int を用いた。(但し、Haskell では環境しだいではもっと大きな数を扱える。)

つまり、もっと大きな数にも対応するなら Integer を使うと良い。ところが、リストの n 番目の要素を取り出す演算 (!!) は、Int 型で定義されているため使えない。

このことの解決策は、二つある。

一つは、型変換を用いる方法。型変換については、Haskell が、強い型付き言語であることを意識してもらうため、授業では触れずにおいた。

この方法では、Int を Integer に変更する (実際は、Num クラスに変換する)。例えば、関数 conv2 の型を

```
conv2 :: Integer -> [String] -> String
```

とする。そして、定義の中で Int にしなくてはならないところで、

```
(digits !! fromIntegral(n))
```

というように、関数 fromIntegral を用いる。(これが型変換である。)

もう一つの方法は、Integer を使ったリストの要素を取り出す関数 (digit) を定義すること。

```
digit :: Integer -> [String] -> String
digit 0 (x:xs) = x
digit n (x:xs) = digit (n-1) xs

conv2 :: Integer -> [String] -> String
conv2 n (s:ss) | n < 10    = digit n digits ++ s
               | m == 0    = conv2 d ss
               | otherwise = conv2 d ss ++ digit m digits ++ s
        where (d, m) = divMod n 10
```

ところで、10 や 100 という数ついて、領収書などでは金額の改ざんを防止する目的で、壱拾や壱百と書く場合もあるようだが、一般には壱拾や壱百とはあまり言わない。(1000 の場合は、壱千とも言うし、単に千と言うこともある。)

このことを考慮したプログラムを考えてみる。少々、姑息な手段ではあるが、例外扱いをする他に方法はないと思われる。

```
digits = ["弐", "参", "四", "五", "六", "七", "八", "九"]
scales = [("", "壱"), ("拾", "拾"), ("百", "百"), ("千", "壱千")]
classes = [ "", "万", "億", "兆", "京"]
```

```
convert :: Integer -> IO ()
convert 0 = putStrLn "金零円"
convert n = putStrLn ("金" ++ conv1 n classes ++ "円")

conv1 :: Integer -> [String] -> String
conv1 _ [] = error "大きすぎる"
conv1 0 cls = ""
conv1 n (cl:cls) | m == 0    = conv1 d cls
        | otherwise = conv1 d cls ++ conv2 m scales ++ cl
         where (d, m) = divMod n 10000

conv2 :: Integer -> [(String, String)] -> String
conv2 0 cls = ""
conv2 n (s:ss) = conv2 d ss ++ digit m s digits
         where (d, m) = divMod n 10

digit :: Integer -> (String, String) -> [String] -> String
digit 0 p ds = ""
digit 1 (r,s) ds = s
digit 2 (r,s) (d:ds) = d++r
digit n p (d:ds) = digit (n-1) p ds
```

さて、最後に「(++) は効率が良くないので、多様すべきではない」ということで、このプログラムからも (++) を排除してみた。

こうすると、digit では、拾と百のときだけ「壱」を使わないという部分がうまく定義できないため、conv2 を各桁を扱う 4 つの関数に分けて定義した。

```
digits = ['壱', '弐', '参', '四', '五', '六', '七', '八', '九']
classes = ['万', '億', '兆', '京']

convert :: Integer -> IO ()
convert 0 = putStrLn "金零円"
convert n = putStrLn ('金' : conv1 d classes (conv2 m "円"))
        where (d, m) = divMod n 10000

conv1 :: Integer -> [Char] -> String -> String
conv1 _ [] s = error "大きすぎる"
conv1 0 cls s = s
```

```
conv1 n (cl:cls) s = conv1 d cls (conv2 m (cl:s))
        where (d, m) = divMod n 10000

conv2 :: Integer -> String -> String
conv2 n s | m == 0    = r
          | otherwise = digit m ('千':r) digits
where m = div n 1000
              r = conv21 (mod n 1000) s

conv21 n s | m == 0    = r
           | m == 1    = '百':r
   | otherwise = digit m ('百':r) digits
where m = div n 100
      r = conv22 (mod n 100) s

conv22 n s | m == 0    = r
           | m == 1    = '拾':r
   | otherwise = digit m ('拾':r) digits
where m = div n 10
      r = conv23 (mod n 10) s

conv23 n r | n == 0    = r
           | otherwise = digit n r digits

digit :: Integer -> String -> [Char] -> String
digit 1 r (d:ds) = (d:r)
digit n r (d:ds) = digit (n-1) r ds
```