7.3 書式を整えよう

さて、循環節の巡回を目にしたのだが、リスト表記はいささか見にくい。この場合はむしろ、改行しながら縦に表示してくれる方がありがたい。そして **Haskell** には、そのための関数が備えられているのだ。

(ghci env.)
```
*Main> mapM print (qsumlist 7)
142857
285714
428571
571428
714285
857142
[(),(),(),(),(),()]
```

見ての通り `qsumlist 7` でリスト表記になるはずの結果を、`mapM print` を用いることで望んだ出力が得られた。でも、最後の `[(),(),(),(),(),()]` って何？

それを説明するには、まず **Haskell** の特性についておさらいをしなくてはいけない。**Haskell** のような純粋関数型言語は、関数が同じ引数で 2 回呼び出されたら異なる結果を返すことは許されない。つまり、関数の状態を変更できないのである。このことは関数が外界から隔離されているに等しい。そのため、キーボードや画面とのやり取りはほとんど不可能である。なぜなら、キーボードや画面は刻一刻状態が変化するからだ。

しかし **Haskell** は、そのような制約の下で外部とやり取りをし、かつ関数の純粋性を保つ工夫がされている。それが、外部とのやり取りをする **I/O アクション**と呼ばれる仕組みである。たとえば `putStrLn` 関数がそれで、

(ghci env.)
```
Prelude> putStrLn "Hello, world!"
Hello, world!
```

のようなことができる。はあ？　それって、たとえば C 言語でいう `printf` 関数と同じもの？って感じがするよね。何しろ `Hello, world!` って、ベタな文字列を表示してるし。でも、ちょっと違うのだ。`putStrLn` 関数は文字列を引数にとることはとるが、受け取った引数を返してはいない。`Hello, world!` を表示しろと命令しただけである。そして、命令された別の関数が `Hello, world!` を表示したのである。

じゃあ、`putStrLn` 関数は何も返さないのかというと、そんなことはなく、裏で「空のタプル」を返している。つまり `()` だ。だから画面に 5 行分の文字列が表示されたとき、空のタプルも 5 個分表

示されるのである。でも、さっき入力したのは putStrLn ではなく print だよね。実は、putStrLn と print は、文字列を出力する点は同じだが微妙に振る舞いが違う。

(ghci env.)

```
Prelude> putStrLn "Hello, world!"
Hello, world!
Prelude> print "Hello, world!"
"Hello, world!"
```

文字列そのものを出力するか、文字列という値を出力するかの違いだ。qsumlist が吐き出すリストは何だっただろうか。それは値のリストだった。だからここで使うのは print の方だ。そしてリストの要素一つひとつを出力したいなら、さらに map する必要がある。しかし、それでは単に要素一つひとつに print 関数が与えられるだけで、出力のための I/O アクションは起こしてくれない。その目的のためには sequence 関数を使わないといけない。結局、いまの場合だと

```
sequence (map print (map (* qsum (quotient [1,7])) [1..6]))
```

と書くことになるだろう。本来の形で見てしまうと「うーむ、( ) の多さはどうにかならんか」と感じるが、実際の入力は sequence (map print (qsumlist 7)) で済む。なんなら**関数適用演算子**である$関数を用いて sequence $ map print $ qsumlist 7 と書けば ( ) は不要だ。その上で、さらに mapM が sequence map を肩代わりしたのだ。

なんとまあ面倒な、と感じたかもしれないね。でも、そういう仕様なのだ。しかしながら、I/O アクションの結果として空のタプルが表示されるのは目障りだと思うでしょ。そのために mapM_がある。これは I/O アクションの結果を表示しない。それを $\frac{1}{13}$ の循環節で試してみよう。

(ghci env.)

```
*Main> mapM_ print (qsumlist 13)
76923
153846
230769
307692
384615
461538
538461
615384
692307
769230
846153
923076
```

空のタプルは現れないね。いい感じだ。だけどちょっと不満がある。1 行目は 076923 と表示してほしいものだ。そうでないと循環節の巡回の様子が正確につかめないからね。

ところが都合のよいことに、**Haskell** には C 言語のように出力書式を指定できる。そのためには Text.Printf モジュールが必要だ。インポートするとプロンプトが*Main Text.Printf> に変わるだろう。これで、出力の書式を望みの形式にできる。

使い方は C 言語とまったく同じと言ってよい。ここでは C 言語には触れないが、書式は"%6d"のように指定して次のように使う。

(ghci env.)
```
*Main Text.Printf> printf "%6d" 123
   123*Main Text.Printf>
```

%6d というのは、6 桁の 10 進数（decimal）という意味である。だから 123 は 6 桁分の表示領域を確保して表示されているのが分かるだろう。6 桁分の表示領域に 3 桁の数値を流したので、左に 3 桁分の空白が見えるね。でも、123 の直後にプロンプトが表示されてしまったのは、私が組版をしくじったのではなく、実際このように表示される。それは、改行がなされていないことが原因だ。改行したいなら、その旨を書式に指定する必要がある。改行は C 言語と同様、\n である。したがって、きちんと表示させるには"%6d\n"と指定するべきであった。

これで桁はきっちりそろうが、やはり 0 も表示させたい。それには 0 の表示を明示するために、書式の指定を"%06d\n"としておくのがよい。これで完璧だ。

(ghci env.)
```
*Main Text.Printf> mapM_ (printf "%06d\n") (qsumlist 13)
076923
153846
230769
307692
384615
461538
538461
615384
692307
769230
846153
923076
```

やったね。ようやく庭の隅っこも整ってきた。これで

(ghci env.)
```
*Main Text.Printf> mapM_ (printf "%0112d\n") (qsumlist 113)
```

なんかを実行しても、きちんと結果を出してくれるんだから。$\frac{1}{113}$ の循環節は、なかなか圧巻じゃないかな。でも、本当に 112 回分の巡回をしているか調べるのは骨が折れるけど。

ところで、mapM_の代わりに mapM を用いると、I/O アクションの結果が表示される。それは当然のごとく空のタプルなのだが、スクリプトは 112 回の循環を処理しているので、相応のものが表示される。実際にやってみれば、あまりにうざったいことに辟易（へきえき）することだろう。