第1週の庭いじり

1.1　自然数から整数への継承

われわれはものを数えるとき

$$1,\ 2,\ 3,\ 4,\ 5,\ \ldots,\ 10,\ \ldots,\ 100,\ \ldots$$

と数えるのが普通である。これらの数はどこまでも大きく数えられるし、どこまでいっても終りというものがない。数学の世界では、ここに登場した数を**自然数**と呼んでいる。

まず、約束がなされたことに気づいただろうか。いま登場した数を自然数と呼ぶ約束のことだ。ここで、何でそう呼ぶの？とか、0 は自然数じゃないの？とかの疑問が浮かぶかもしれないね。数学では “何々のことを何々と決める” ということが頻繁にでてくる。そしてその決め方が自然と納得できるものもあれば、場合によってはどうしてそう決めるのか不思議に思うことがあるだろう。約束—すなわち**定義**—というものは、大体の雰囲気で決める場合もあれば、深い意味があってそう決める場合がある。君たちが戸惑うのは、深い意味があって定義されることがらだろう。その場合は、定義に納得いかないこともあるはずだが、深い意味があるだけに当面は謎のままになってしまうものだ。その定義に関する内容を深く理解したとき、謎が氷解することが往々にしてあるので、楽しみはとっておくのがよいだろう。

自然数という呼び名は、大体の雰囲気から妥当と思われる。人間が自然発生的に使い出した数が 1, 2, 3, . . . だから。

ちょっと待ってほしい。われわれが現在、自然に使う数字は 0, 1, 2, 3, . . . ではないか。それなら 0 を含めて自然数と呼ぶほうが自然だろう。こう考える人はいるかな。たしかに、そのとおり。それは間違った考えではない。ただ、ここでは習慣に従い、1 から数える数を自然数であると定義しておこう。あとで分かるように、0 は特別扱いしたい数だから。

ところでわれわれが普段使う数には -1, -5 などもある。このような数は 0 より小さい数を表すために作られた数だ。自然界には 0 より小さいものはないと言ってよいだろう。何もない状態が 0 の状態だから、もうこれ以上なくなる状態はありえないのだ。ところが人々は 0 より小さい状態を

概念として作ってしまった。借金や気温や水深などはそのよい例になっている。

0 より小さい数は、自然数に “–” の**符号**を付けて表している。1 や 5 は、0 より 1 や 5 だけ大きい数ととらえるならば、−1 や −5 は、0 より 1 や 5 だけ小さい数ととらえることができる。数に “–” の符号をつけて、0 より小さい数を表すことは新たな約束となる。しかし、ここでは約束もさることながら、数の継承がなされたことに注意を払ってほしいのだ。

もし、負の数を新たに定義するだけなら a, b, c, ... という “数” を作れば済む。ところが実際は、自然数を土台に −1, −2, −3, ... と数を拡張している。つまり自然数を継承したわけだ。継承するということは、自然数の持つ性質も受け継ぐことを意味している。たとえば数の間隔は、負の数でも 1 ずつだし、大きい数字を使うほど 0 から離れた数になることなどがそうだ。

結局、数は 0 を中心にして

$$\ldots,\ -10,\ \ldots,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ \ldots,\ 10,\ \ldots$$

と並んでいることになる。そしてこれらの数を**整数**と呼び、さまざまな数の基準として使うことにするわけである。

それではこの辺で、**Haskell** に何かやってもらおう。庭いじりの最中に書くのは、ほんの短いスクリプトである。数学の話題にはなるべく立ち入って話したいけれど、スクリプトの説明はほどほどで済ませている。理由は、数学の考え方のほうを重視しているからだ。もしスクリプトが難しいと感じるなら、それは言語の理解が不足しているためではない。問題解決のための知識が不足しているのだ。

では **Haskell** を起動させよう。the Glasgow Haskell Compiler（GHC）をインストールした場合は、Windows ならコマンドプロンプトに、Macintosh（または Linux）ならターミナルウィンドウに、“`ghci`” と打ち込む。すると、画面には

```
Prelude>
```

というプロンプトが表示される。この状態は「対話モード」と呼ばれる環境で、簡便な使い方ができる。しばらくは対話モードで進み、適当なところでプログラミングを記述して実行する環境に移っていこう。

まず、簡単な計算なら対話モードで数式を入力すれば答を出してくれる。四則計算を行うための**演算子**（`+`, `-`, `*`, `/`）の優先順は数学で慣れ親しんでいる通り、乗除（`*`, `/`）が加減（`+`, `-`）より優先される。べき乗—すなわち**指数**計算（`^`を用いる）—はさらに乗除より優先される。計算順を変えるには ( ) を使うことも数学の規則通りだ。あとでたっぷり計算するけれど、とりあえず整数を使って簡単に計算規則を確認しておこう。

(ghci env.)

```
Prelude> 1+2*3
7
Prelude> (1+2)*3
9
Prelude> 1+2/3
1.6666666666666665
Prelude> (1+2+3+4+5+6)/3
7.0
```

はじめの 2 例を見れば数学の規則を踏襲しているのが分かるはずだ。

割り算は基本的に実数値で答が返ってくる。ただし、1+2/3 の末尾がちょっと変なのはコンピュータが 2 進数で計算していることが原因である。この点は、あとで説明するつもりだ。また 4 例目のように、たとえ整数だけの計算であっても、除算の計算結果は実数扱いになる。**Haskell** は整数値の 7 と実数値の 7.0 を厳密に区別するのだ。

べき計算では、扱える整数値に上限・下限の制限はない。だから、びっくりするような桁数をこなしてくれる。普通の言語では 2^{1000} など、軽く**オーバーフロー**してしまうのに、**Haskell** はメモリの許す限りきちんと結果を出してくれるのだ。

(ghci env.)

```
Prelude> 2^1000
10715086071862673209484250490600018105614048117055336074437503883703510511
24936122493198378815695858127594672917553146825187145285692314043598457757
46985748039345677748242309854210746050623711418779541821530464749835819412
67398767559165543946077062914571196477686542167660429831652624386837205668
069376
```

2^{1000} は 302 桁の整数になるので、コンピュータのウインドウサイズに合わせて改行されて表示されるはずだ。さらに平方根の計算も簡単にできる。

(ghci env.)

```
Prelude> sqrt 2+1
2.414213562373095
Prelude> sqrt(2+1)
1.7320508075688772
```

sqrt は演算子と考えてもよいが、実際は**関数**である。関数は四則・べき計算より優先度が高い。sqrt 2+1 は 2+1 が接近して見えたとしても sqrt 2 が先である。よって sqrt 2+1 は $\sqrt{2}+1$ のことである。2+1 を優先したければ ( ) を付ければよい。これで sqrt(2+1) は $\sqrt{2+1}$ を意味する。

もし、sqrt 2 の優先をはっきりさせたければ (sqrt 2)+1 と書いてもよい。しかし、**Haskell** では ( ) を付けない記述は多く見られるので、sqrt 2+1 の計算順を正しく見られるように慣れる

べきだろう。場合によっては空白を利用して `sqrt 2 + 1` と書いてもよいのだから。

また、**Haskell** は式の**真・偽**の判定、つまり式が正しいか間違っているかの判定もできる。

(ghci env.)

```
Prelude> 1+2+4 == 8
False
Prelude> 1+2+4 /= 8
True
```

$1+2+4$ は 8 に等しくないので `False`（偽）が返る。**Haskell** では “等しい” ことを表す記号は “`==`” である。“`=`” は式の定義など、別の意味で使われる。初めは間違えやすいので気をつけよう。“等しくない” ことを表す記号は “`/=`” である。たしかに $1+2+4$ は 8 に等しくないので `True`（真）が返っている。

このように、**Haskell** は自由度の高い言語であるものの、ある部分では非常に気難しい面もある。いまの入出力例を見ると、ちょっと気難しい面が垣間見えるはずだ。今回の庭いじりでは気難しいことは省いておこう。まあ、それでも多少のプログラミング作法は身に付くだろうから。それに何が気難しいかは、ほんの一握りの好奇心があれば誰にでも体験できる。いろいろ試してみればよいからだ。ただ、好奇心は満たされても不満が募るかもしれない。不満の解消は各自で調べてもらうしかないけれど。