0. . . の夢見鳥

0.1 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, . . ., 10, . . ., 100, . . ., 1000, . . .,』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。

さて、夢見鳥となって翔び始めたばかりだが、いま『　』内に列挙した数字の列は単にそのままタイピングしたのではない。『　』の数字の列は『

```
\begin{luacode*}
    for i = 1, 5 do tex.print(i .. ', ') end
    tex.print('$\\dots$, ')
    n = 1
    for i = 1, 3 do
        n = n * 10
        tex.print(n .. ', $\\dots$,')
    end
\end{luacode*}
```

』のようにして出力されたものである[1)]。馬鹿やってんじゃないよと思ってもらって結構だけど、これが Lua の仕事なのだ。Lua の記述のしかたは大抵のスクリプト言語とそう変わるところはない。私の知る範囲で説明しておこう。

まずスクリプトを書いて Lua に処理してもらうためには、基本的に `luacode*`環境を使うのがよい。この環境に書かれた内容を Lua が理解し処理をするのである。

実際の Lua の命令である 1 行目、

```
for i = 1, 5 do tex.print(i .. ', ') end
```

は 1 から 5 までの出力である。Lua の場合は `for` で指定した回数分、`do`〜`end` にはさまれた記述を繰り返す。この場合は、**変数** `i` に与えた数値から 5 回の繰り返しであることが分かり、`i` は `do` のあとに続く `tex.print` 文の中にも使われている。出力は TEX の仕事なので `tex.print()` と書く。

1） TeXShop (version 4.51) を使用しての結果。

( ) 内の書式は、変数をそのまま書けば変数に格納された値が出力される。**文字列**を出力したければ ' ' または " " で囲めばよい。.. は出力される数値・文字を連結する**演算子**であるから、この場合は i が 1 から 5 まで変化しながら ,␣（␣ は 1 個の空白の意味）と連結されて出力されるのである。ここまでで『1, 2, 3, 4, 5, 』だ。

次に ..., を出力したいのだが、tex.print() 内で \ を TEX の命令と認識させるには、\ を**エスケープ**し \\ と書く必要がある。だから \\dots と書いたのだ。TEX の命令である \dots はそのままなら …を、数式モードで $\dots$ とすれば ... を出力する。

続けて『10, ..., 100, ..., 1000, ..., 』を表示したいのだが、for 文は改行しながら適当に**インデント**を入れて書くほうが見やすいのでそうした。ここでは n = 1 で**初期化**し、n = n * 10 としながら for〜end ループで 4 行を使っているが、10^i を用いて

```
for i = 1, 3 do tex.print(10^i) end
```

と書けば 1 行で済むと思ったかもしれない。だったら実際、試してみよう。すると『10.0, ..., 100.0, ..., 1000.0, ..., 』になってしまう。^ は整数値だけを与えても実数値を返す仕様なのである。ここには不向きであった。

さあ、もとの路に戻ろう。自然数が登場したんだったね。そのとき約束がなされたことに気づいただろうか。『1, 2, 3, 4, 5, ... 』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方が自然だろう。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。実際、集合論で自然数を構成するときは**空集合**を基準にし、それを数 0 に対応させている。ただ、ここでは習慣にしたがい 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**をつけて表している。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『$\dots,\ -10,\ \dots,\ -5,\ -4,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ \dots,\ 10,\ \dots$』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。たったいま書き並べた整数は、実は『

```
\begin{luacode*}
    tex.print('$\\dots,\\ -10,\\ \\dots,$')
    for i = -5, 5 do
        tex.print('$' .. i .. '$, ')
    end
    tex.print('$\\dots,\\ 10,\\ \\dots$')
\end{luacode*}
```

』と書いて出力したわけだが、`tex.print` 文が長いのはまとめ書きをしたためである。Lua を通さず TEX で -10 前後の部分を記述すれば、`$\dots$`, `$-10$`, `$\dots$`, とするか `$\dots,\␣-10,\␣\dots,$` とするところである。数式モードでは `\␣` を明示しないと空白が詰まってしまう。繰り返しになるが、`tex.print()` 内の `\\␣` は改行を意味する TEX の命令ではなく、エスケープさ

れた \␣、すなわち空白 1 個であることに注意されたい。

それより −5 から 5 までの出力が

```
tex.print('$' .. i .. '$, ')
```

となっていることに注目してほしい。本当は `tex.print(i .. ', ')` でもよかったのだが、TEX で負の数を表す場合は数式モード、すなわち `$$` で囲む必要がある。さもないと -5 などという無様な出力になってしまう（実際に `tex.print(i)` を試すとよい）。だったら `tex.print($i$)` で良さそうなものだが、`$` は TEX のシンボルであって Lua のシンボルではないので結局エラーになる。だから `$` は Lua に文字として出力させ、変数 `i` と連結したところで TEX に −5 を認識させている。ま、気難しいけど慣れればどうということはない。

大雑把に言って `luacode*`環境の中では、Lua の命令にしたがって文字列を作り替えているようなものだ。いまの例では

最初の `tex.print`文が	`$\dots,\␣-10,\␣\dots,$`	を、
続く `for`文が	`$-5$,␣$-4$,␣`(途中略)`␣$4$,␣$5$,␣`	を、
最後の `tex.print`文が	`$\dots,\␣10,\␣\dots$`	を作り、全体として

『`$\dots,␣-10,␣\dots$,␣$-5$,␣$-4$,␣`(途中略)`␣$4$,␣$5$,␣$\dots,␣10,␣\dots$␣`』へ展開されて、TEX が処理をして『$\cdots,\ -10,\ \ldots,\ -5,\ -4,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ \ldots,\ 10,\ \ldots$』となるのである。

このとき `for` 文の直前、すなわち `$\dots$` と `$-5$` の間に ␣（空白）が入るように記述されてなかったと思う。しかし、TEX の規則では改行は空白 1 個であり、また空白は何個連続しても 1 個に相当するはずだったので、適切なアキができたものと考えてよいだろう。Lua の正確な挙動について、本当のところは知らないんだけどね。

このように TEX と Lua を組み合わせて組版をすると、単に組版をする以上の処理が可能であるものの、ある部分では非常に気難しい面もある。いまの出力例が本当のところどんな処理をしたのかを知りたければ、LuaTEX についてもっと学ばなければならないだろう。私は気難しいことに細かく言及するつもりはないし、言及できるほど Lau に精通しているわけではない。ここは LuaTEX を正しく習得する路ではないのだ。宙をひらひら舞う夢見鳥のように見えて、実際はふらふらと危なっかしい路を進んでいるのである。だから側溝に落ちてもよい覚悟で翔んでほしい。