0. . . の蝶道

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

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

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。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 $-1, -2, -3, \ldots$ 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。

さて、ここで TEX のマクロについて触れておこう。マクロとは自前の命令のことで、マクロを定義しておけば、以降同じ処理をする際はマクロをひとつ記述すれば済む。たとえば

```
\newcommand\macroA{\TeX のマクロ例（ソースコード）}
```

のように、`\newcommand` 命令を用いて`\macroA` という名前の命令を定義し、『これは`\macroA`␣である。』と書けば『これは TEX のマクロ例（ソースコード）である。』が出力される。“␣” は英字空白 1 個を意味し、この空白がないと “`\macroA` である” までが命令の名称であると勘違いされ、“Undefined control sequence.”（命令が未定義である旨のメッセージ）が返されてしまう。なぜなら TEX は、英字空白などで命令の区切りを判断するからである。以後、“␣” は明示しないので注意されたい。

引数（ひきすう）を取るマクロを定義することもできる。引数の数は

```
\newcommand\macroB[2]{\TeX のマクロ例#1（ソースコード#2）}
```

のように、命令名の後ろに [ ] で示し、引数が使われる位置に`#1`, `#2`, ... をあてがう。いまの例なら、『これは`\macroB{1}{1-A}` である。』と書けば『これは TEX のマクロ例 1（ソースコード 1-A）である。』が出力される。つまり、“`\macroB`” が “`\TeX` のマクロ例`#1`（ソースコード`#2`）” に置き

換わる際に “#1” が “1” に、“#2” が “1-A” に置き換えられ、一緒に “\TeX” は “TEX” に展開される仕組みである。

もう少し例を続けよう。10 以下の正の偶数を

```
\newcommand\gusuTen{$2$, $4$, $6$, $8$, $10$}
```

で定義しておくと、『10 以下の正の偶数は\gusuTen である。』と書けば『10 以下の正の偶数は 2, 4, 6, 8, 10 である。』が出力される。しかし、これでは面白みに欠けるし融通も利かない。そこで、引数を用いて

```
\newcommand\gusu[1]{#1 以下の正の偶数は$2$, $4$, \ldots, #1 である。}
```

で定義しておくと、『\gusu{10}』と書けば『10 以下の正の偶数は 2, 4, . . . , 10 である。』が出力されるし、『\gusu{50}』と書けば『50 以下の正の偶数は 2, 4, . . . , 50 である。』が出力される。余談ながら#1 の直後に空白は要らない。引数は#9 までしか利用できないので、#の次は 1 文字と決まっているからだ。

また、マクロ\gusu は “. . . ” を用いてごまかしたことに注意してもらいたい。ところが、このようなごまかしが常に有効である保証はない。3 の倍数や 8 の倍数などというときは、3, 6, 9, . . . や 8, 16, 24, . . . と書く必要があり、その都度異なる 3 個の引数を与えるマクロを定義しなくてはならない。3 個の数だけを出力するマクロに 3 個の引数を与えるのでは本末転倒である。倍数は先頭の数さえ分かれば、あとは 2 倍、3 倍、. . . するだけなので、先頭の数だけを引数にしたマクロが書ければ申し分ない。それには、次のようなマクロ定義がよいだろう。

```
\newcommand\baisu[1]{%
  \newcount\n
  \n=#1
  \number\n, \advance\n#1 \number\n, \advance\n#1 \number\n, \ldots
}
```

すると、『3 の倍数は\baisu{3}である。』と書けば『3 の倍数は 3, 6, 9, . . . である。』が、『8 の倍数は\baisu{8}である。』と書けば『8 の倍数は 8, 16, 24, . . . である。』がそれぞれ出力される。マクロを見やすくするために、適宜、改行や空白を入れてあるが、そのために余計な空白が出力されることがある。その場合は、行末の適所に “%” を入れたり、空白を詰めたりして調整しなくてはならない。実際、1 行目に “%” がないと 3 や 8 の前に空白が入る。

TEX では、命令直後の “␣” と命令でない文字列直後の “␣” では意味が異なる。命令直後の “␣” は単に区切りの意味として、文字列直後の “␣” は文字の続きの空白として扱われる。ただ、アル

ファベット以外の**1 文字命令**直後の “␣” は空白扱いになる。たとえば$記号を出力する 1 文字命令 “\$” は、『\$␣99』と書くと『$ 99』が出力される。空白を入れたくなければ『\$99』と書く。TEX の空白の扱いは、マクロの中でも頭を悩ませるかもしれないが、多少のことはあまり気にしないほうがよいかもしれない。

では、マクロ\baisu の説明をしておこう。まず、新たにマクロを定義するときは\newcommand 命令を使う。この場合は、1 個の引数を取る\baisu という名前のマクロが定義され、定義内容が “{” から “}” までの間に記述される。したがって、\baisu が実際に行う処理は、2–4 行目に記述されている 3 文である。2 行目は**カウンタレジスタ**（以下、道中では**変数**と呼ぶ）の定義で、整数変数は\newcount 命令で行う。小数値を扱いたいときは別の命令があり、あとで登場する。変数は “\” で始まる文字列で命名し、大文字と小文字は区別される。つまり、\var と\VAR は異なる変数である。

3 行目の\n=#1 は、変数\n に引数である#1 の値を代入している。続く\number\n の\number は、\n を 10 進数で表した数値に**展開する**—すなわち\n の値を出力する—命令である。したがって、この命令によって\n に代入されている値が出力される。\baisu{3}を実行した場合、#1 として取り込まれた値 3 が\n に代入され、まず “3,” が出力されるわけである。マクロ中の “,” は、多くのプログラミング言語にありがちな “命令の区切り” ではなく、単に出力される “文字” であることに注意されたい。TEX は命令等の終端を空白で判断するが、“,” や “\” などの特別な文字があれば、そこが終端または区切りと判断する。

次の\advance は加算命令で、\advance x a と書けば、x に a を加えたものを新たな x の値とするものである。一般的なプログラミング言語ならば、\advance x a はおそらく x = x+a と書かれる。したがって、\advance\n#1 は\n に#1 の値が加算されることになる。倍数は 2 倍、3 倍、...と増えるが、それは次々と数を加算することに等しいので加算命令を使った。

そのあとは見て分かるように、同じことを 2 回繰り返して、最後に “...” を出力させればよい。

マクロを見て何か気づいただろうか。TEX のマクロは、基本的に文字列を置き換える。たとえば、

```
\newcommand\A{おはようございます。青木です。}
```

というマクロを定義して『\A さて、...』と書いたら、その文章は『おはようございます。青木です。さて、...』に置き換わる。しかし、\baisu{3}はそのまま置き換わったわけではない。純粋に置き換わったのは “\number\n,” にくっついている “,” だけである。\number\n は置き換わったというより、展開されたものが出力されているのである。また、\newcount\n や\n=#1 などは展開されるわけでも出力されるわけでもない。

要するにこういうことだ。TEX におけるマクロというものは、基本的に文字列を置き換えているのだが、あるものは展開されるべきものだったり、また、あるものは単に処理がなされるだけのものであったり、と様々な形態のものが混在しているのである。

このように、TEX のマクロはプログラミング言語と同等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。いまの出力例を見ると、ちょっと気難しい面が垣間見えるはずだ。今回は気難しいことに細かく言及するつもりはない。ここは TEX のマクロを正しく習得する道ではないのだから。あえて言えば、宙をひらひら舞う蝶が進む道—蝶道（ちょうどう）とでも言おうか—のように、見るからに危なっかしい道なのだ。