3.3 大きい桁数の扱い

TEX では何百桁もの計算をすることは難しい。ここまでの蝶道でも、極端に大きな整数や小数点以下何桁もの計算はしてこなかった。でも、TEX だって他のプログラミング言語のような計算はしてきたじゃないか。だったら TEX 以外のプログラミング言語は、どうやって長大な桁数の計算をしているのだろう。

コンピュータプログラムは普通、大きな数を扱う場合**配列**を利用して計算する。しかし配列を用いなくても、メモリが許す限り、大きな桁数が扱える言語も数多くある。残念なことに TEX にはそのような仕組みがない。それでも、基本的に扱える数に限界、それも数桁程度の数で限界になってしまうようなとき、どんな工夫で何百桁もの計算をするのか知るのは意義あることだと思う。

まず、TEX は数字を並べるだけなら何桁でも並べられる。たとえば

```
\newcount\c
\loop \ifnum\c<20 \number\c
\advance\c1 \repeat
```

を処理すれば 012345678910111213141516171819 が出力される。30 桁の数である。いや、ちょっと待って。先頭が 0 の数って変だよね。そう、これは “数” ではなく “文字列” だから計算に使えない。

ならば、次のようなものはどうだろう。

```
\newcount\Ma \newcount\Mb \newcount\Mc
\Ma=1111 \Mb=2222 \Mc=3333
\newcount\Na \newcount\Nb \newcount\Nc
\Na=7777 \Nb=5555 \Nc=3333
$\number\Ma\number\Mb\number\Mc+\number\Na\number\Nb\number\Nc
= \advance\Ma\Na\advance\Mb\Nb\advance\Mc\Nc
\number\Ma\number\Mb\number\Mc$
```

これを処理すると $111122223333 + 777755553333 = 888877776666$ である。見て！　12 桁どうしの足し算が`\advance` を使ってできている。でも、すぐにメッキは剥がれるよね。これでは繰り上がりが出たらアウト！

とは言うもののこれは大きなヒントである。それは、繰り上がりの処理ができれば`\advance` で計算できるということなんだから。実は他のプログラミング言語でも同じことをしている。複数の変数をつなげて足したり掛けたりしているのだ。ただし、もちろん繰り上がりなどの処理をするんだけど。

繰り上がりの仕組みを考えてみよう。たとえば $22.55 + 11.77 = 34.32$。小数値だけれど、22.55 は $A = 22$、$a = 55$ という 2 変数を $A.a$ で表していると思ってほしい。同様に、11.77 は $B = 11$、$b = 77$ という 2 変数を $B.b$ で表しているということだ。したがって、実際は $A+B = 22+11 = 33$、

$a+b=55+77=132$ になってしまうが、$(A+B)$ と $(a+b)$ を小数点でつなげた数は 33.132 でなく 34.32 でなければならない。何をする必要があるかというと、132（1 百 3 十 2）の 1 百を、単なる 1 として上の桁に加算しなくてはいけない。この操作は計算上

$$132 \div 100 \text{ の}\underline{\text{商 1}}\text{と、}\underline{\text{余り 32}}\quad \text{を分離して}$$

上の桁へ加算していることと同じであある。商や余りを求める操作というのは、一般的なプログラミング言語では簡単にできることである。大抵は命令ひとつで済む話であることが多い。したがって、さほど面倒な操作ではないといえる。

以上のことから、`A.a+B.b = A.a` の一般のアルゴリズムは次のようになるだろう。もちろん 2 桁の数を例にとった場合であり、`A.a+B.b = A.a` の表記は`\advance\A\B` の動作を意識している。

a) `A` ← `A+B`； `a` ← `a+b`

b) `q` ← `a/100` の商； `r` ← `a/100` の余り

c) `A` ← `A+q`； `a` ← `r`

d) `A.a` を出力

このアルゴリズムのよいところは、繰り上がりがなくても問題ないことである。繰り上がりがなければ `q` は `0` であり、また `r` は単に `a` であり、それは繰り上がりのない `(A+B).(a+b)` を意味する。

引き算ならどうなるだろう。$22.55-11.77=10.78$ を例にとってみる。小数点で数が分離されているというものの、それぞれ $22-11$、$55-77$ ではうまくない。桁の貸し借りによって、$21-11=10$、$155-77=78$ として小数点でつなぐのである。

すると `A.a-B.b = A.a` の一般のアルゴリズムは次のようになるだろう。

a) `a < b` であれば：

b) `A` ← `A-1`； `a` ← `a+100`

c) `A` ← `A-B`； `a` ← `a-b`

d) `a < b` でなければ：

e) `A` ← `A-B`； `a` ← `a-b`

f) `A.a` を出力

引き算の場合は引いて負の値になっては困るので、あらかじめそれを判定しておきたい。桁の貸し借りがあれば A の 1 を a に 100 として貸し付けて（←とんでもない高利貸しだな）引く。そうでなければ単に引く。しかし、これでは計算のたびに判定が必要なので、少し工夫が要るだろう。

掛け算は順繰りに掛ける手もあるが、整数どうしの掛け算ならば単に繰り返し足す方が手間がかからないはずだ。割り算も繰り返しの引き算としてもよいだろうが、むしろ順繰りの割り算と、余りの繰り越しにする方がよいかもしれない。その場合は次のようになるだろう。ABC/p = abc の例だと思って見てほしい。

a) a ← A/p の商； r ← A/p の余り

b) B ← B+r*100

c) b ← B/p の商； r ← B/p の余り

d) C ← C+r*100

e) c ← C/p の商

f) abc を出力

最後に C/p の余りを r に代入していないが、たとえば四捨五入などのために必要になると思えば代入しておくとよいだろう。

いずれにせよ大きな整数の四則計算は、少しだけ手間をかければ正確にできるのである。そして、TEX でもできないことではない。その証拠は見てもらうのが一番だが、最初に示したマクロに手を加えても仕方ないので、再びこの道に戻ったとき、出力の仕方を工夫するのと一緒に、長大な桁数を持つ数の計算をすることにしよう。