1.3 コラッツの問題

有名な数列はフィボナッチ数列だけではない。フィボナッチ数列は不思議な数列だが、他にも未解決の数列がある。それは次の規則で作られる。

はじめに勝手な自然数を用意する。次の項は「前の項が偶数なら 2 で割り、奇数なら 3 倍して 1 を足す」というものだ。たとえば 50 から始めると

$$50,\ 25,\ 76,\ 38,\ 19,\ 58,\ 29,\ 88,\ 44,\ 22,\ 11,\ 34,\ \ldots$$

のような数列が出来上がる。数列はどこまでも続くように思える。しかし、そうではないのだ。簡単なマクロで試してみよう。

```
\newcommand\collatz[1]{%
  \newcount\n \newcount\c
  \n=#1 \number\n\
  \loop
    \ifodd\n \multiply\n3 \advance\n1 \else \divide\n2 \fi
    \number\n\
    \advance\c1
  \ifnum\c<20 \repeat
  ...
}
```

これで、『50 から始まるコラッツ[1]数は`\collatz{50}`である。』と書けば『50 から始まるコラッツ数は 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 ... である。』が出力される。

ここではコンピュータプログラムにおける、重要な分岐処理を理解してほしい。コンピュータは繰り返し同じことをするのに苦痛を感じないが、思い通りの処理を自動的にしてくれるほど気が利いているわけではない。コラッツの問題と呼ばれるこの数列の行方を調べるには、偶数と奇数の判定ができなくてはならない。

ところで、ここからはマクロが引数を取るようにしてみよう。いままではマクロの中に初期値を記述していたので、別の値から始めたいときは初期値を書き直す必要があった。使い勝手のよいマクロにするには、マクロが引数を取るようにしておく方がよいだろう。そこで変数`\n` にはマクロ中で値を与えず、引数で得た値を代入するようにした。直後に`\number\n\` で`\n` の値を出力させたのは、このあとの処理で`\n` の値が変化してしまうからである。

さて、TEX には奇数であることを判定する便利な命令`\ifodd` がある。これを`\ifodd\n` のように使うと、`\ifodd` は`\n` が奇数のときに**真**を返す。一般に`\if` 系の命令は、真ならば`\if` の次に記

[1] ローター・コラッツ（1910–1990）：ドイツの数学者。

述された処理が行われ、**偽**ならば \else 後に記述された処理が行われる。\if 命令の終りは\fi である。見ての通り\if を逆さまに読んでいる。

するとマクロは\ifodd 命令の 1 文で、\n が奇数なら $3n+1$ を、偶数なら $\frac{n}{2}$ を求めて出力していることが分かるだろう。\n\ につけた “\ ” は、空白を入れて出力させるためのお約束だったね。そして新たな値となった\n は、\loop〜\repeat 命令によって\c<20 まで処理されるのである。繰り返しに入る前に先頭の\n を出力したため、出力される項は 21 項になる。

マクロは、最後におまけのように “...” を加えてみた。この先も数が続く意味で用いたのだが、一般的なプログラミング言語では、こういうものを出力するために print 文などを使うだろう。しかし、TEX は書いた通りに出力される方が自然なので、こんなことができるという見本である。

いろいろな数に対して、コラッツの問題の操作を繰り返せば、結果の予想がつくだろう。その予想は正しい。しかしコラッツの問題は予想であって証明ではない。現在でも、どうやら確からしいという程度の予測はできても、未だ証明は得られていない。ところで繰り返しの範囲を\c<20 のように区切ってしまうと、コラッツの操作が終了する前にマクロが終わることもある。先の例はそうなっている。それでは困る。そこで範囲を決めるのではなく、\n が 1 になったところでマクロが終了するように改良しておこう。

改良のついでと言ってはなんだが、\loop〜\repet の代わりに\@whilenum 命令を使ってみよう。

```
\makeatletter
\newcommand\collatzB[1]{%
  \newcount\n
  \n=#1 \number\n\
  \@whilenum\n>1 \do{%
    \ifodd\n \multiply\n3 \advance\n1 \else \divide\n2 \fi
    \number\n\ }%
}\makeatother
```

これで、『50 から始まるコラッツ数は\collatzB{50}である。』と書けば『50 から始まるコラッツ数は 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 である。』が出力される。

やっていることは前のものと同じだが、違いが 2 つある。ひとつ目の違いである\@whilenum 命令は、直後に書かれた条件が満たされる限り、\do{〜}で定義された処理がいつまでも続く。つまり\@whilenum 命令は\do ブロックと一緒に使われる。処理がブロック内に書かれるため\repeat は不要である。マクロの終了は、条件\n>1 が満たされなくなったとき、すなわち\n が 1 になったところで\do ブロックが終わるということだ。\do ブロックの最後が “}%” であることに注意しておこう。\do は命令なのだが、{ }内の文はルーティンのかたまりであって命令ではない。そのた

め “}” 後の改行は空白 1 個に相等することになる。もしここに “%” がないと、\do ブロックが終了したときに空白が 1 個出力されるのだ。マクロの実行には問題ないけれど、ちょっと気を遣ってみた。

2 つ目の違いは、ほんのわずかなことであるが、とても重要なことでもある。\do ブロック内で \number\n を用いて\n の値を出力するとき、空白を空けるために “\ ” を付けているね。もしここが\number\n\ }でなく、\number\n\}だったらマクロは正常に動作しない。“\}” になってしまうと、これは文字 “}” を意味することになる。すると、\do ブロックの閉じ括弧がないことになってエラーを起こすのだ。どんなプログラミング言語でも 1 文字の違いでエラーを引き起こすものだが、TEX のマクロではエラーの原因は少し独特である。

また、このマクロには特殊文字 “@” が含まれているので、\makeatletter と\makeatother が必須なのは言うまでもない。

引数に 27 を与えるとなかなか楽しめる。『\collatzB{27}』は以下のようになる。

『27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 』

また、コラッツの問題はちょっとした拡張ができる。前の項が偶数なら 2 で割り、奇数なら 3 倍して 1 を足すという規則を、たとえば奇数なら 5 倍して 3 を引く、みたいにしてみることだ。これは\multiply\n3 \advance\n1 を\multiply\n5 \advance\n-3 に変えるだけなので、すぐに実行に移せるだろう。もちろん他の規則にしてもよいし、偶数の規則を変えたって構わない。

さて、いくつか試してみることを勧める。ところが実際にやってみると、コラッツの問題ほどうまくハマらないことが分かるだろう。すぐに循環したり、いつまでたっても収束の気配を見せなかったりとね。なぜ、$3n+1$ だとうまく運ぶのだろうか。このことは未だ謎なのだ。もっともらしい理由はあることはあるのだが、決定的な説明ではない。興味をひいたら取り組んでみるとよいだろう。