1. . . の蝶道

1.1　フィボナッチ数列

数列とは、数が何らかの規則で並んでいるものを言う。

$$0,\ 2,\ 4,\ 6,\ 8,\ 10,\ \ldots$$

は偶数の数列だが、偶数列を作る規則は 0 から始めて「前の**項**に 2 を加える」というものである。

さて、ここに 1, 1 から始まる数列がある。次の項を作る規則は「直前の 2 項の和」である。すなわち 1, 1 の次に来る項は 2 となり、数列は 1, 1, 2 と延びる。規則を繰り返し当てはめれば、次の項は 3 で数列は 1, 1, 2, 3 と延び、さらに次の項は 5 である。これが延々となされ

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ \ldots$$

なる数列が出来上がる。このようにしてできる数列を**フィボナッチ**[1]**数列**と呼ぶ。

フィボナッチ数列を TEX のマクロで再現してみよう。

```
\newcommand\fibonacci{%
  \newcount\a \newcount\b \newcount\t \newcount\c
  \a=1 \b=1
  1 1 \c=3
  \loop
    \advance\a\b
    \t=\a \a=\b \b=\t
    \number\b\
    \advance\c1
  \ifnum\c<21 \repeat
}
```

これで、『フィボナッチ数列の最初の 20 項は`\fibonacci` である。』と書けば『フィボナッチ数列の最初の 20 項は 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 である。』が出力される。

[1] フィボナッチ（1174?–1250?）：イタリアの数学者。本名、ピサのレオナルドの通称。

マクロはまず、はじめの 2 項を\a と\b として、それぞれに 1 と 1 を代入している。\t は値を退避するための変数、\c はカウンタ用の変数である。そして、繰り返しのために\loop〜\repeat 命令が登場しているのはいままで通りだ。

マクロ中に不自然に書かれたように見える 1 1 は、そのまま出力される。TEX のマクロの中にあっても、単なる文字列はそのまま出力されるので、実はこっちが自然で、\advance などの命令が不自然なのだ。文字を組版している最中にマクロが実行されている、と言った方が正しいかもしれない。1, 1 が出力されたあとには\c=3 がある。この代入は\a、\b と同じ行で行えば済むことだが、1, 1 を出力したらカウンタは 3 になっているはずだ。このことを意識して、この位置に書いた。20 項分のフィボナッチ数を出したいため、最後の判定が\c<21 なのも出力される項数に合わせたものだ。しかし、\c は単にカウントするだけの変数だから、\c=0 から始めて\c<20 と判定しても同じである。このほうが\c に値を代入することもなく、また<20 という数値からループを 20 回通ることが分かりやすいかもしれない。

フィボナッチ数列の計算の仕方から、\advance\a\b と書けば次の項が計算されて、それが\a に格納されるのは分かるだろう。ここで\a の値を出力すれば次の項が出力されるはずだが、同時に\a と\b の値を変えておきたい。なぜなら、この時点での\a の値は\b の次に来る数になってしまったので、\a、\b の順に大きくなるように並べたいからだ。

それには、いま\b であった値を\a に、\advance\a\b で求めた値を\b に繰り延べなくてはならない。それが\t=\a \a=\b \b=\t である。このとき、いきなり\a=\b とやってしまうと、せっかく求めた次の項が上書きされてしまう。それを回避するために\t が必要だったのだ。これで\b が出力したい次の項になった。あとは、その繰り返しになる。

退避用の変数を用いない方法もある。変数の数を増やしたくなければ次のようにするのもよい。

```
\newcommand\fibonacciB{%
  \newcount\a \newcount\b \newcount\c
  \a=1 \b=1
  1 1 \c=3
  \loop
    \advance\b\a
    \advance\a-\b \a=-\a
    \number\b\
    \advance\c1
  \ifnum\c<21 \repeat
}
```

これで『\fibonacciB』を処理すれば、出力は前とまったく同じ、以下のようになる。

『1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 』

でも、このマクロは何をしているのだろう。\advance\b\a は先のマクロとは\a と\b の順序が逆だが、2 項の和であることは間違いない。ただ、新しい値が\b に代入されただけだ。\b だって？\b は次の項の和にも使うはずだから上書きされたら困るだろう。でも、安心してほしい。\b は元の\b の値を保持している。新しい b' は $b' = a + b$ だから元の b を含んでいるだろう。このあとで\advance\a-\b を計算すれば、新しい a' は $a' = a - b' = a - (a + b) = -b$ となって元の b —ただし正負が逆だけど—が代入される。そして\a=-\a とすれば $a' = b$ だ。これでめでたく 2 項が玉突きされたのだ。

さて、マクロでは 20 項分の数列を出力したに過ぎない。もし、ずーっと先まで見たければ\c の範囲を大きくすればよい。すると初項から目的のところまでのフィボナッチ数が列挙される。しかし、それでは煩わしいので n 番目のフィボナッチ数だけを出力するマクロに直しておくのも悪くない。それには、\loop〜\repeat の中にある\number\b\を外に出してやればよい。そのとき、最初に書いた 1 1 は取り除いておこう。これがあると、いつだって 1 1 が出力されてしまうから。

```
\newcommand\fibonacciC{%
  \newcount\a \newcount\b \newcount\c
  \a=1 \b=1 \c=3
  \loop
    \advance\b\a
    \advance\a-\b \a=-\a
    \advance\c1
  \ifnum\c<41 \repeat
  \number\b
}
```

これで、『40 番目のフィボナッチ数は\fibonacciC である。』と書けば『40 番目のフィボナッチ数は 102334155 である。』が出力される。

\c はもう少しだけ大きな範囲にできるけれど、さらに大きな範囲を入力しないように。TEX はエラーこそ起こさないものの、妙な値を表示するはずである。何が起こるかは実際に試すとよいだろう。そのようなことが起こるのは TEX 特有の現象ではない。大抵のプログラミング言語なら似たようなことが起こる。その理由は、しばらく先の適切な場所で話すことにしよう。いまは居心地がよいとは言えない道を這い回っているだけに、変なことが起きてもあわててはいけない。