6. . . の蝶道

6.1　完全数

6 は興味深い数のひとつだ。それはどういうことかというと、6 の約数は 1, 2, 3, 6 だが、

$$1 + 2 + 3 = 6$$

となっている。この性質を満たす数は 6 の近辺にはない。たとえば 8 は、約数として 1, 2, 4, 8 を持つが、明らかに $1 + 2 + 4 \neq 8$ である。約数をもれなく取り出すことができれば、6 の次にこの性質を満たす数が 28 であることを知るのは容易（たやす）い。実際、28 の約数は 1, 2, 4, 7, 14, 28 であるから

$$1 + 2 + 4 + 7 + 14 = 28$$

になっている。この性質を持つ数を**完全数**と呼ぶ。

性質を誤解のないように言えば、完全数とは

自分自身を除く約数の総和が自分自身に等しい数

であると言える。なんと、日本語にするほうが式で説明するより難しくなってないかい？　数学とはそんなものだ。

それなら、28 の次の完全数はいくつだろうか。ちょっと計算すればすぐに見つかりそうだ。素数のように、明らかに完全数になり得ないものは省ける。約数の数が少ないものや (素数)^n のような数もたいてい省けそうである。このように候補となる数を絞っていけば、意外に早く見つかるかもしれない。悪いね。そんな簡単には見つからないのだ。よほど根気よくなければ、次の完全数を見つけるのは難しい。根気が勝負になるときこそコンピュータプログラムの出番というわけだ。本当は TEX の出番ではないけれど、無理矢理登場してもらおう。

いきなり完全数を求めるマクロを書くのも大変だろうから、まずは約数を見つけるマクロから見ていこう。

```
\newcommand\measures[1]{%
  \newcount\n \newcount\d
  \d=1
  \loop
    \n=#1 \divide\n\d \multiply\n\d \advance\n-#1
    \ifnum\n=0 \number\d, \fi
  \advance\d1 \ifnum\d<#1 \repeat
  \number\d
}
```

これで、『45 の約数は\measures{45}である。』と書けば『45 の約数は 1, 3, 5, 9, 15, 45 である。』が出力される。

いままでの知識さえあれば十分理解できるマクロだ。コードは与えられた数に対して、約数をすべて出力することになっている。ただし、とんでもなく無駄が多いマクロである。

\loop〜\repeat 命令の中で除数\d の値を 1 から始め、1 ずつ増やしながら割っていくことになっているが、整数が 1 で割れるのは当たり前なので、はっきり言って無駄な行為である。しかし、1 で割らずに単に出力するだけでも 1 行使うのだし、どのみち\d には 1 からにしても 2 からにしても、何らかの数の代入が必要なので、無駄と知りながらそうしたのだ。また、約数を求める行為は素数を求める行為と違うので、2 で割れても 4 で割れるか試す必要があるし、1 ずつ大きな数で割る必要もある。ただし、$\frac{n}{2}$ までの数で試せばよい。それより大きい数が約数になることはないのだから。しかし、またもや無駄に割っている。このマクロは引数で得た値の直前まで割り算をする。なぜそうしたかと言うと、$\frac{n}{2}$ を判定するのが単に面倒だったからだ。大規模なソフトウェアを作ってるわけではないので、ここでは効率は無視した。

\loop を抜ける判断は\d<#1 で行っている。実は、この判断を最初の方で行えば最後の\number\d は必要なかった。これがあるのは、約数を “,” と一緒に出力すると、最後の約数の後にも “,” が付いてしまう。これを避けたかったので、こうなったのだ。

マクロは約数を列挙するものだが、素数を見つける役にも立つ。なぜなら、素数は 1 と自分自身しか約数を持たない数だから、たとえば\measures{17}に対しては 1 と 17 だけが出力される。

ところで、もっと効率的なアルゴリズムにするにはどうすればよいだろう。当然のことながら、たとえばある数 n が d で割れたなら、その商である $\frac{n}{d}$ も約数である。つまり約数は基本的に 2 つ同時に見つけることができる。基本的と表現したのは、49 が 7 で割れたからと言って、$\frac{49}{7}$ が別の約数とは言えないからだ。しかし、除数 d と商 $\frac{n}{d}$ を一緒に出力してしまえば、割り算は $\sqrt{n}$ まで試せばよいことになる。これは以前、素数を求める際に指摘した手法だし、計算量を減らす効果も十分だ。

だが、完全数を求めるためには、これではまったく不十分なのだ。あとで分かるように、完全数は素数とは較べものにならないくらい稀にしか現れない。つまり、計算量が減ること自体は喜ばしいけれど、結局ほとんどの数で調査が不発に終わる。効果的に調べるには、計算量を減らすのではなく、調査しなくてもよい数を飛ばすことである。

さて、約数をすべて列挙する関数が書けたので、それらの和をとれば完全数を探すことができる。

```
\makeatletter
\newcommand\pnfind[1]{%
  \newcount\n \newcount\d \newcount\m \newcount\sum
  \m=6
  \@whilenum\m<#1 \do{\d=1 \sum=0
  \loop
    \n=\m \divide\n\d \multiply\n\d \advance\n-\m
    \ifnum\n=0 \advance\sum\d \fi
  \advance\d1 \ifnum\d<\m \repeat
  \ifnum\sum=\m \number\m, \fi
  \advance\m1}%
}\makeatother
```

これで、『1000 未満の完全数は`\pnfind{1000}`である。』と書けば『1000 未満の完全数は 6, 28, 496, である。』が出力される。最後に “,” が付いているのは見苦しいだろうが、本当はマクロそのものが見苦しいのだ。

マクロは関数`\measures` に、和を求めるために変数`\sum` を追加したものである。`\sum` は自分自身以外の約数の和を加算する変数だ。最初の完全数は 6 であることが分かっているので、`\m=6` から始めている。この 6 はマクロ`\measures` では引数として`#1` に与えられたが、マクロ`\pnfind` の引数は完全数候補の上限として使われるため、別の変数`\m` に代入して約数を調べている。約数を見つけるルーティンは、`\measures` で見つけた約数を出力する代わりに、変数`\sum` に加算している。そして、調査される数`\m` と約数の合計が等しくなったら、それが完全数である。

ところで、マクロには`\@whilenum` と`\loop`〜`\repeat` が**入れ子**になって使われているだろう。ここは`\loop`〜`\repeat` を入れ子にしても同じである。ただし、その場合は注意が必要だ。単に入れ子にしただけでは`\loop`〜`\repeat` が互いに干渉してしまい、期待した動作をしてくれない。そうしないために、内側の`\loop`〜`\repeat` は`{}`を用いてグルーピングしなくてはならない。TEXbook に掲載してあったマクロ`\primes` を振り返ってみよう。`\testprimality` が`{{...}}`で定義されていたはずだ。`\testprimality` は`\primes` において、`\loop`〜`\repeat` の内側で実行される仕様なので、`\loop`〜`\repeat` が互いに干渉しないようにするためである。

TEX がこのマクロを処理したときに分かるように、出力までには少し時間を要したはずだ。原因のひとつは、約数を見つけるルーティンに無駄が多いままだからだ。それに加えて完全数は非常

に少ない。そのため、4 番目の完全数を探すためには相当の時間を覚悟しなくてはならない。それもそのはずで、完全数は $2^{n-1}(2^n-1)$ の形をしている。いま探した 3 つの数でちょっと確認してほしい。これでは指数関数的に大きな数になってしまうから、たとえ完全数がたくさんあったとしても、気軽に発見できないのだ。

そこで忠告をしておこう。このマクロで 4 番目の完全数を出力させようと思わないように。その行為は暴挙以外の何ものでもない。4 番目の完全数は 4 桁の数だが、TEX の守備範囲だと考えないでほしい。このアルゴリズムは、3 桁の 496 を見つけるのでさえ少々時間を要する。4 桁の数なら割り算を試す回数が 10 倍になり、なおかつ調べる数の総数も 10 倍になっているから、単純に考えても計算時間は 100 倍だ。かりに 1000 までの調査が 1 秒で済んでも、10000 までの調査には 100 秒（1 分 40 秒）はかかることになる。待ち時間としては微妙だ。

結局のところ、コンピュータの計算速度を過信してはいけないということだ。そのために、どうしても人の手で、効率的なアルゴリズムが必要になるのである。効率のことは、もうしばらく先で考えよう。もし、手っ取り早く 4 番目までの完全数を知りたいなら`\pnfind`を修正するのでなく、$2^{n-1}(2^n-1)$ の形の数だけ調査するマクロを書けばよい。驚くほど早く結果を目にすることができる。でも、TEX で完全数を見つけること自体、常軌を逸しているんだけどね。