4. . . の蝶道

4.1 ゴールドバッハの予想

4 はちょっとだけ面白い性質を持つ。それは

$$4 = 2 + 2 = 2 \times 2$$

のように同じ数どうしで、和と積に分解できる唯一の自然数だからである。さらに使われている数 2 は、唯一の偶素数でもある。数学は “唯一” という言葉が好きである。

さて、ここで**素数**という言葉が登場した。そもそも私がこの蝶道を這い回るはめになったのは、TEXbook[1]に素数を出力するマクロが見本として載っていたからである。TEXって数式をきれいに組版するだけじゃなかったんだ。そのマクロを掲載しておこう。ただし、改行や空白を適宜入れてある。

```
\newif\ifprime \newif\ifunknown
\newcount\n \newcount\p \newcount\d \newcount\a
\def\primes#1{2,~3%
  \n=#1 \advance\n by-2 \p=5
  \loop \ifnum\n>0 \printifprime \advance\p by2 \repeat}
\def\printp{, \ifnum\n=1 and~\fi \number\p \advance\n by-1}
\def\printifprime{\testprimality \ifprime \printp \fi}
\def\testprimality{{\d=3 \global\primetrue
  \loop \trialdivision \ifunknown \advance\d by2 \repeat}}
\def\trialdivision{\a=\p \divide\a by\d
  \ifnum\a>\d \unknowntrue \else \unknownfalse \fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse \unknownfalse \fi}
```

これで、『素数を小さい順に 100 個並べると`\primes{100}`である。』と書けば『素数を小さい順に 100 個並べると 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421,

[1] TEX ブック（Donald E.Knuth 著、鷺谷好輝訳、アスキー出版局）

431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, and 541 である。』が出力されるのである。こりゃあ、びっくりだ。驚いてばかりいられないので、蝶道に戻ろう。マクロの解析はそのあとでする。

一般に自然数は、素数の積に**一意的に**分解できる。一意的とは、掛け算の順番を無視すれば、ただ一通りの積にしかならないことを意味している。しかし自然数は、素数の和に一意的には分解できない。それは

$$10 = 5 + 5 = 3 + 7 = 2 + 3 + 5$$

を見れば分かるように、いくつかの分解の仕方がある。

素因数分解のように一意的に解が求められるものは、分かりやすいし研究の対象にもなりやすい。一方、素数の和にする場合はつかみどころがない。どちらかと言えば、面白味に欠ける話題だ。と思ってはいけない。実は大変面白いものがあるのだ。

ゴールドバッハ[2]はオイラー[3]との手紙のやり取りで、「2 以上の偶数は 2 つの素数の和になる」ことを話題にした。当時の 2 人は、1 を素数の仲間に入れていた様子なので、$2 = 1 + 1$ も話題に含まれていたようだ。しかし、われわれは 1 を素数の仲間に入れていないので

4 以上の偶数は 2 つの素数の和になる

と言うべきだろう。現在、この問題に対する**反例**も証明も見つかっていない。

惜しいことに、この中には唯一という単語は含まれない。つまり、4 以上の偶数はただ一組の 2 つの素数の和になる、わけではない。実際、20 は $20 = 3 + 17 = 7 + 13$ のように 2 組の素数の和になっている。

ゴールドバッハの予想を検証するには、素数の判定ができなくては困る。そこで素数を判別するマクロが必要となる。さっきのマクロを使えばよいのだが、それは素数を順に出力するものだったので、ある数が素数かどうか判定するには不向きである。そこで、変更すべきところをちょいと変えてみよう。

```
\def\primeornot#1{\ifodd#1 \p=#1\printifprimeB \else 合成数\fi}
\def\printifprimeB{\testprimality \ifprime 素数\else 合成数\fi}
```

これで、『9 は`\primeornot{9}`で、17 は`\primeornot{17}`である。』と書けば『9 は合成数で、17 は素数である。』が出力される。なんとわずか 2 行のマクロだ。TEXbook の見本では、マクロ定義に`\def` を用いていたのでそのままにした。近頃では`\newcommand` で定義することが一般的の

[2]クリスティアン・ゴールドバッハ（1690–1764）：ロシアのドイツ人数学者。
[3]レオンハルト・オイラー（1707–1783）：スイスの数学者・物理学者。

ようだが、どちらでも同じである。ただ、\newcommnad が引数を [2] のように個数で指定するのに対し、\def は直接#1#2 と指定する。

変更は一部だけなので少し不都合なところがある。もとのマクロは素数を出力する際、2, 3 だけは判定抜きに出力している。また、その後は 5 以降の奇数だけを調査している。そこで引数が偶数なら、はなから\primeornot は引数を合成数であると判断する。そのせいで\primeornot{2}が合成数と判定されてしまう。逆に、\primes は 3 を素数判定から除外しているせいで、\primeornot は 3 も合成数と判定してしまう。もちろん、引数が 2 か 3 のときは素数と判定するように記述を追加すればよいのだが、とりあえず変更を少なくするために目をつぶった。

それでは、もとの\primes が何をしているか見ておこう。

はじめに\newif という命令が新しい。\newif は真偽を保持する変数を定義する命令だ。ここでは、\ifprime、\ifunknown が定義されている。\ifprime を例にとると、変数に当たる\ifprime は\primetrue か\primefalse のいずれかの値を持つ。このことは、\ifprime ひとつで判定文になっていることを意味する。補足説明はあとでしよう。

マクロの本体は\primes#1 であり、わずか 3 行のマクロでしかない。\primes が行っているのは、まず 2, 3 を書き出し引数を\n に代入する。この\n が素数の個数を数えることになるので、すでに 2, 3 を出力してしまったために 2 を減じている。そして、最初に調べる素数候補は 5 であり、それを変数\p に代入している。\p が送られる先では、\n がまだ正の値のとき—すなわち調べるべき数があるとき—\printifprime を実行し\p を次の候補である 2 つ先の数に改めること、その繰り返しである。ここで Knuth 氏は\advance\n by-2 のように、加算する数を by 付きで記述している。さすが英語を母語とするだけに自然な感じがする。不思議なことに\by という命令じゃないのに by は出力されない。何でこういう仕様なんだろう？

ところで、たったこれだけのマクロが必要な素数をすべて出力するのは、マクロ本体が面倒な作業を下請けに出しているからである。下請け先は\printifprime に他ならない。\printifprime の仕事は、\p が素数かどうか判定し素数なら出力する、というものだ。こいつは要領の良いことに、素数の判定はさらなる下請けである\testprimality に委ね、あろうことか出力作業まで下請けの\printp に回しているのだ。

では、先に\printp がすることを見ておこう。\printp は素数の出力専門である。したがって、ここに送られてくる\p は間違いなく素数である。よって、最後の 1 個を受け取ったら体裁のために and を追加するが、それ以外では\p を出力し素数カウンタを 1 減らすのである。

いちばんの面倒を請け負ったのは\testprimality だ。これは受け取った\p を素数テストにか

ける。素数テストは 3 から 2 ずつ大きくした奇数で割り続けることだが、実はこの作業の受け持ちは`\trialdivision`なのである。

`\trialdivision`は`\p`を受け取るものの、`\p`を割り算テストにかけて値を変えてしまうわけにはいかない。そこで、自由に割って構わない変数`\a`に代入する。実は TEX には剰余を求めるための命令がない。それがあれば余りが 0 のとき割り切れたと判断できるんだけどね。そこでどうするかというと、再度`\a`に除数`\d`を掛けて、もとに戻れば割り切れたと判断するのだ。そしてこのときに`\p`の値が必要になる。で、割り切れないときが素数で、割れたときが素数でないことになって、`\ifunknown`に`\unknowntrue`か`\unknownfalse`が格納されるのである。

このようにマクロは、下請けに次ぐ下請けに仕事を回しているため、複雑さを増しているように感じるだろう。しかし、このようにするのが良いのだ。理由は、それぞれのマクロが短いため見やすくなることと、分割されたマクロが関数として再利用しやすくなるためである。