4.2 素数の出力

マクロ\primeornot は、入力された数が素数かどうか判定するものだった。それなら初めから素数の一覧表があると便利だろう。マクロ\primes は素数の個数を与えて出力したので、最後の素数が予測しづらい。それより、与えられた n までの素数をもれなく出力する方が、必要な素数が分かってよいかもしれない。\primes を改良—いや、改変かも—してみよう。

```
\newif\ifprime \newif\ifunknown
\newcount\p \newcount\d \newcount\a
\def\nprimes#1{2,~3%
  \p=5
  \loop \ifnum\p<#1 \printifprime \advance\p by2 \repeat}
\def\printp{, \number\p}
\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、『50 未満の素数は\nprimes{50}である。』と書けば『50 未満の素数は 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47 である。』が出力される。

マクロの変更はごく一部に止まる。引数を与えたとき、引数以下までの素数と解釈させるか、引数未満の素数と解釈させるかで終了判定が微妙に違う。\loop 内の\ifnum の判定が、素数の個数でなく素数の値になったので、\n ではなく\p を調べればよい。このとき、“≦” に相当する演算子がないため、引数未満の素数と解釈させている。もし、引数以下の素数と解釈させたければ、\ifnum\p<#1 \printifprime ... は\ifnum\p>#1 \else \printifprime ... と記述すればよいだろう。本当は\p<=#1 のような演算子があればよいのだろうが、なくても何とかなるものだ。新しいプログラミング言語は洗練されているかもしれないが、パズルみたいにプログラムを組む機会があってもよい。

他には、\n による判定が不要になったので、\newcount\n をはじめとして、\n に関わる処理は削除した。そして、変更はこれだけである。元になったマクロのように、処理が細分化されていると変更の影響が広範囲に及ばないのが良い点だ。

ところで素数の調べ方は 1 通りではない。別の調べ方についても話しておこう。素数は 2 を除いてすべて奇数だから、3 以上の奇数の中から、約数を持つ数を順にはねれば素数が残る。約数を持つ数は逆に言えば素数の倍数だから、素数の倍数を先にはねれば素数が残る。このアルゴリズムは

エラトステネス[1]の篩（ふるい）と呼ばれる。

ところで、ここに示した `primes` 関数は、計算効率の観点からは無駄が多いマクロである。それは、たとえば 997 が素数かどうか調べるとき、997 の手前までの全部の奇数で割り算をしている。実際は $\sqrt{991} \approx 31.48$ より小さい素数で割れなければ、それより大きな数で割れっこないのだ。この差は大きい。だから、適切な場所で`\p` が引数の平方根より小さいことを判定して、真なら次の数を試すようにするとよい。ただ、引数の平方根を計算するには専用のマクロを書かないといけないし、精度が心配である。そこで、このようなときは`\p` と引数の平方根を比較するのではなく、`\p` の平方と引数を比較するのがよい。これなら精度の心配は要らない。あとで参考になる例を示すつもりだ。

エラトステネスの篩を用いてマクロを書いたらどうなるだろう。エラトステネスの篩のアルゴリズムは次のとおりだ。

a) n を 2 とする

b) n を素数として確定する（○で囲む）

c) n の倍数をすべて消去する

d) 残っている数で、いちばん小さい数を n とする → b) へ戻る

実は、エラトステネスの篩は大変優れたアルゴリズムである。それは、このアルゴリズムで 100 までの素数を調べた一覧を見てもらえば分かる。

	②	③	~~4~~	⑤	~~6~~	⑦	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

[1]エラトステネス（275B.C.–194B.C.）：ギリシア人の数学・天文学者。

一覧は、7 を素数として確定し、7 の倍数をすべて消したところまでの状況である。次は 11 に○をつけて、11 の倍数を消していく手はずであるが、もうその必要はないのだ。なぜなら、n の倍数を消すことと、消される運命の数を n で割ることは同じことだからだ。すなわち、100 までの素数を調べるための割り算は $\sqrt{100} = 10$ までででよい。したがって、アルゴリズムが 11 に到達したところで、100 までの素数がすべて見つかっているのだ（消えずに残った 25 個が素数）。

それなら、これをマクロにすれば大変効率的に素数を求められる、と誰もが思うかもしれない。つまり、2 から n までのリストを用意して、2 の倍数から順にリストから除けばよいと考えられるからだ。ところで、TEX ではリストをどのように用意するというのか。かりに用意できても削除すべき数の位置を特定するのは不可能ではないにしても、相当に困難だと思われる。リスト方式は TEX では現実的ではない。結局のところ、アルゴリズムの効率は、必ずしもマクロを書く効率と同じではないということだ。

ちなみにアルゴリズムの語源は、代数学の父と称されるアラビアの数学者、アブー・アブドゥッラー・ムハンマド・イブン=ムーサー・アル=フワーリズミー[2]の名前の最後が変じたものと言われている。ほんまかいな？

[2] アル=フワリズミー（800?–850?）：バグダードで活躍した数学・天文学者。