4.2 素数の出力

関数 pmsg(n) は、入力された数が素数かどうか判定するものだった。それならはじめから素数の一覧表があると便利だろう。ひとくちに一覧表と言っても、m 個分の素数の一覧表と n までの素数の一覧表では内容が異なる。ここでは、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。そこで関数を『

```
\begin{luacode*}
function primes(n)
    pL = {2, 3}  key = 2
    for i = 5, n, 2 do
        for j = 1, key do
            if i % pL[j] == 0 then
                goto exit
            end
        end
        table.insert(pL, i)
        key = key + 1
    ::exit::
    end
    return pL
end
\end{luacode*}
```

』をのように定義して、『\directlua{tex.print(primes(100))}』とすれば『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』が出力される。

スクリプトを説明する前に、素数の調べ方について話しておこう。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。スクリプトはエラトステネス[1)]の篩（ふるい）とは少し違うが、似たような方法で素数を探している。ここでは、3 から順にすべての奇数をテストにかける。テストは、その奇数より小さい素数で順に割ってみるだけである（奇数を割るので素数 2 で割ることはしない)。割れてしまえばその奇数ははねられ、割れなければ素数のリストに追加する、というものだ。

では、スクリプトの説明だ。まず、pL = {2, 3} に注目してほしい。これはテーブルを利用して素数のリストを作成するための種である。{} の代入でテーブルが作られることは前に言ったけど、

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

その際初期値を与えることができる。この場合は、最初から素数 2, 3 を含んだテーブルが作られたことになる。テーブル名の pL は primesList の略だ。

テーブルに 2 と 3 を代入したのは、素数が 2, 3 から始まることと、割り算テストに使われる最初の素数が 2, 3 だからだ。つまり、この時点でテーブルには pL[1] = 2, pL[2] = 3 が格納されている。

素数は 3 までリスト入りしているので、次に調べる数は 5 からで、その後 2 ずつ増やした数が調査の候補となる。候補となった数がテーブルにある素数で割れるかどうか調べるのが、二つ目の for ブロックである。ここで問題出現だ。あとのコードで説明するが、このスクリプトはテーブルに随時素数を追加する仕様なので、テーブルの要素数が変化する。そこでテーブルの現在の要素数を示す変数 key を利用した。

もし候補の数 i がリスト内のすべての数で割れれば、候補の数は用無しである。そのときは、goto 文で指定先の ::exit:: へ飛ぶ。goto 文を悪く言う人は一定数いるようだが、この場合は二つ目の for ループを抜けたいのだ。if 文を抜けるだけでは table.insert(pL, i) が実行されてしまうのでまずい。

候補の数が素数でなければ、goto 文によって二つ目の for ブロックを抜けるが、ここはまだ一つ目の for ループの中である。したがって次の候補が調査対象となる。ここでも if i % pL[j] == 0 のチェックを受けるが、最後まで乗り切ればそれは素数である。table.insert(pL, i) はこのときはじめて実行される。テーブル pL に要素 i を追加するときにこう書く。こうして最後までリストに追加すれば、リストを表示すればよい。return 文でテーブルを丸ごと返せばよい。

ところで、ここに示した primes 関数は、計算効率の観点からは大変よくないスクリプトである。それは、たとえば 997 が素数かどうか調べるとき、リストにある 991 までの全部の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、スクリプトの適切な場所で i > math.sqrt(n) を判定して、真ならループを抜ける一文を付け加えるとよい。簡単なことだから、君たちの課題にしておこう。

とこれで、いまテーブルを利用して素数を一覧にしたのだが、テーブルは本来**連想配列**として使うようである。連想配列とは

```
pTable = {name = 'Teffko', age = '45', friend = 'Lua'}
```

のように格納して、name で 'Teffko' が参照できるものである。だから

```
for key, value in pairs(pTable)
```

と書いて、テーブルから = で関連づけされたペアをまとめて変数 key や value に取り出せる。これを利用すると関数 prime(n) は

```
\begin{luacode*}
function primes2(n)
    pL = {2, 3}
    for i = 5, n, 2 do
        for key, p in ipairs(pL) do
            if i % p == 0 then
                goto exit
            end
        end
        table.insert(pL, i)
    ::exit::
    end
    return pL
end
\end{luacode*}
```

と書き直せる。key の値は使わないが、その代わり ipairs() を用いて pL[1] から番号順に取り出すようにしている。

素数の一覧を得るには、エラトステネスの篩を用いてスクリプトを書くこともできる。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

エラトステネスの篩が大変優れたアルゴリズムであることは、100 までの素数を調べた一覧を見てもらえば分かる。一覧表は Lua の仕事だけど省略するね。10×10 の表を作りながら、素数チェックをして◯、/ を書き加えただけなので、いまならたぶん君たちにもできるだろう。

1	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~~

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、これをスクリプト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。ところが、リストから数を削除し続けるのは簡単なことではない。削除操作自体は簡単なのだが、削除によって数の順番が詰まってくるために、削除すべき数の位置を特定するのは不可能ではないが、相応に困難だと思われる。結局のところアルゴリズムの効率は、必ずしもスクリプト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。