7.3 長桁数の加・減・除

あ、ここは以前訪れた場所だ。そのとき見過ごしたところを翔び回ってみよう。円周率の詳細な値を見たいんだ。前に通った路で $\frac{1}{239^m}$ の値を精確に計算した方法は円周率の計算に使えそうだ。その前に、マチンの式に最小限必要な、長桁数を加・減・除する関数『

```
\begin{luacode*}
function disp(aL) -- テスト出力用
    for i = 0, #aL do
        tex.print(string.format('%08d', aL[i]))
    end
end

function addL(aL, bL)
    aL[0] = 0
    for i = #aL, 1, -1 do
        aL[i-1] = aL[i-1] + (aL[i] + bL[i]) // 100000000
        aL[i] = (aL[i] + bL[i]) % 100000000
    end
    disp(aL) -- テスト出力用
end

function subL(aL, bL)
    aL[0] = 0
    for i = #aL, 1, -1 do
        aL[i-1] = (aL[i-1] - 1) + (100000000 + aL[i] - bL[i]) // 100000000
        aL[i] = (100000000 + aL[i] - bL[i]) % 100000000
    end
    disp(aL) -- テスト出力用
end

function divL(aL, b)
    aL[0] = 0  l = #aL  aL[l+1] = 0
    for i = 1, l do
        aL[i+1] = aL[i+1] + (aL[i] % b) * 100000000
        aL[i] = aL[i] // b
    end
    disp(aL) -- テスト出力用
end
\end{luacode*}
```

』を作っておこう。乗算は省略するね。今回は必要ないから。

ここでは長桁整数を aL, bL と書くことにする。マチンの式で必要な計算は aL ± bL と aL ÷ b の計算である。b は長桁数ではない整数である。

まず、addL 関数は長桁数の足し算用で、長桁数の引数を 2 個とる。最初に aL[0] = 0 があるのは、繰り上がりのためのダミーである。ここでは長桁数 aL を、テーブルの各要素をつなげて

(aL[0])aL[1]aL[2]aL[3]...aL[#aL]

という大きな桁の数になっていることを想定している。# はテーブルのサイズを求めるための修飾子で、#aL 書いてテーブル aL のサイズを取得できる。for 文を見て分かるとおり、末尾の桁 aL[#aL] から最上位の桁 aL[1] までの各桁で足し算の処理をするので、長桁整数は長さは可変でよい。i の値は減るので for i = #aL, 1, -1 なのである。

for ループ内の 2 行が足し算のルーティンである。1 行目が繰り上がり処理で 2 行目が実際の足し算になっていることが分かるだろうか。ちなみに、テーブルの各要素は 8 桁の数値を格納することにしているので、100000000 が登場した。

本来は return 文でテーブルを丸ごと返すのだけれど、ここでは確認のための表示に差し替えてある。このとき、8 桁未満の数であっても 0 を表示しないとことには意味をなさないので、そのための書式を用いた。Lua が出力する書式はいろいろな指定ができるので研究しておこう。

subL 関数は引き算である。足し算とほぼ同じ考え方であるが、引き算ルーティンの処理は気を使うところだ。引いたとき負の値にならないように、あらかじめ上の桁から 1 を 100000000 にして借りていることに注意してほしい。

割り算の divL 関数は以前と同じ仕様だ。割る数は長桁数ではなく、単に整数である。ここではちょっとパズルみたいなことに悩まされる。ダミーのためにテーブルの（要素数）+ 1 が必要なのは以前と変わらない。しかし、もし

```
aL[#aL+1] = 0
for i = 1, #aL do
    aL[i+1] = ...
```

のように書いてしまうと、たとえば長整数 aL[0]...aL[5] を想定した場合、aL[#aL+1] = 0 は aL[6] = 0 である。ところが、ここで aL[6] ができると for i = 1, #aL は for i = 1, 6 となり、次の aL[i+1] は i = 6 のとき aL[7] を指してしまう。するとインデックスが超過してエラー

となる。`l = #aL` へ退避させたのはその対策であった。

さて、加・減・除関数の説明はこのぐらいにして、動作を確認しておこう。

『`\directlua{addL({67891234, 30000000, 55555555}, {33111111, 77777777, 44444445})}`

$= 00000001\ 01002346\ 07777778\ 00000000$』

『`\directlua{subL({67891234, 30000000, 55555555}, {33111111, 77777777, 44444445})}`

$= 00000000\ 34780122\ 52222223\ 11111110$』

『`\directlua{divL({100000000, 00000000, 00000000}, 239)}`

$= 00000000\ 00418410\ 04184100\ 41841004\ 4400000000$』

`addL` と `subL` は

$$\begin{array}{rrrr} & 67891234 & 30000000 & 55555555 \\ \pm) & 33111111 & 77777777 & 44444445 \\ \hline \end{array}$$

の計算である。繰り上がりがあるので `aL[0]` の値も出力した。合っているね。

`divL` は以前やった $\frac{1}{239}$ の計算だ。割られる数は 1.0 を模しているので、計算結果は先頭の `aL[0]` が 0. だと思ってほしい。これも合っているね。末尾の `aL[#aL]` はダミーなのでゴミが入っている。

もし乗算用に `mulL(aL, b)` を作るなら、それは `addL` 関数の `+` を `*` に変える程度でよい。それは君たち自身でやってみることを勧める。