2.4 べき乗の計算

さあ、べき乗の計算でかなり大きな数が現れても、桁の処理をうまくやれば何とかなることが分かった。さっそく 2^x あたりの計算に利用してみよう。しかし、いくら大きな桁を扱えるといっても所詮コンピュータのことである。ある程度の限界というものはある。例えば 2^{1000} まで扱うためには、何桁の数が扱えなければならないのだろう。

ここで指数の考えが登場する。10^1 が 2 桁の数、10^2 が 3 桁の数、10^3 が 4 桁の数、...とくれば、10^x が $(x+1)$ 桁の数であることは容易に想像がつくというものだ。正確には、10^1 から 2 桁の数が始まる、と言うべきだろう。そのため、$10^{1.1}$, $10^{1.2}$, $10^{1.3}$, ...,$10^{1.9}$ などはすべて 2 桁の数であり、10^2 となったところから 3 桁の数が始まるのだ。

従って 2^{1000} が何桁の数であるかを知るには

$$2^{1000} = 10^x$$

を解いて、x の値を知ればよいことになる。この方程式は容易に解けるものではないが、対数を使えばその限りではない。両辺について**底** 10 の対数をとれば、

$$1000 \log_{10} 2 = x$$

であるが、$\log_{10} 2 \approx 0.30103$ であることを使って $x \approx 301.03$ を知ることになる。つまり 2^{1000} は 302 桁の整数なのだ。

よーし、それならここでは、400 桁までの数が扱えるようにがんばってみよう。と言いたいところだが、説明の都合で 40 桁までにさせてもらおう。マクロは 2^{100} の計算例だ。

programming list [PowerOf2.vba]

```
 1 : Sub PowerOf2()
 2 : Dim p(11), n, i, j As Integer
 3 : Range("A2:J2").NumberFormat = "0000"
 4 :
 5 :     p(0) = 1
 6 :     For i = 1 To 9
 7 :         p(i) = 0
 8 :     Next i
 9 :
10 :     For n = 1 To 100
11 :         For i = 9 To 0 Step -1
12 :             p(i + 1) = p(i + 1) + (p(i) * 2) \ 10000
13 :             p(i) = (p(i) * 2) Mod 10000
14 :         Next i
15 :     Next n
16 :
17 :     j = 0
```

```
18 :     For i = 9 To 0 Step -1
19 :         j = j + 1: Sheet1.Cells(2, j) = p(i)
20 :     Next i
21 : End Sub
```

まず、見なれない記号がいくつか現れたね。このマクロは、A2 セルから J2 セルまでを利用して 2^{100} を表示することにしている。そこで 3: 行目でその指定をしているのだが、詳しくは後で説明しよう。

2: 行目で用意した配列変数の説明をしておこう。ここで `p(0)` 〜 `p(10)` までの 11 個の入れ物を準備したわけだが、実は `p(10)` の入れ物はダミーである。なぜダミーが要るかはすぐ後で分かる。さて、5: 行目で `p(0) = 1` を代入した以外は、6: 〜 8: 行目で `p(1)` から `p(9)` まで `0` で初期化したことになる—実はあえて `0` で初期化しなくとも、配列を宣言したときに `0` が代入されている—が、これはこういうことだ。用意した配列は、`p(0)` がいちばん下の桁で `p(9)` がいちばん上の桁を想定している。つまりこの時点で、配列を `p(9)`...`p(0)` と並べると、0...01 なる数ができたことになる。0...01 は何桁の数だろうか。配列の変数を 10 個用意したから 10 桁の数？　残念でした。後からの説明にも関係するのだが、これは 40 桁の数になる。なぜなら各 `p(i)` には 4 桁の数が与えられるからである。

`p(i)` に 4 桁の数が与えられるのには理由がある。`Integer` 型の整数が 2 バイトだからだ。2 バイトの整数は数万の大きさで頭打ちになってしまう。このことは 9999 までは保証されるが 99999 までは保証されないことを意味する。よって 9999 まで保証されるものを 10 個つなげれば、40 桁の数が保証されるのである。

10: 行目の `For` 文で、`n` を 100 回繰り返しているが、これは続く 11: 〜 14: 行の `For-Next` のブロック—これはすべての `p(i)` を 2 倍することに相当する—の 100 回の繰り返しである。これで 2^{100} を計算しているわけだ。

このマクロの主要部分は、11: 行目からの `For-Next` 構文にある。ここは `For-Next` 構文であるが、たった 1 回だけ 40 桁の数を 2 倍するところだ。単純に各 `p(i)` を 2 倍しただけでは、繰り上がりを無視することになってしまう。

そのため 12: 行目の `(p(i) * 2) \ 10000` により、繰り上がりがあった場合に、上の桁—ここでは `p(i + 1)` を上の桁に想定している—に繰り上がった数を加えている。記号 `\` は割り算の商—もちろん整数値だ—を求める演算子[1]で、`A \ B` と書けば `A` を `B` で割った商のみ求められる。

ところで `For` 文が `i = 9` から代入を始めていることに気付いているかい？　繰り上がりの関係で `i = 0` から始めるとまずいのだ。なぜなら `i = 0` から始めると、せっかくの繰り上がりが上書きされてしまうからだ。

[1] JIS キーボードでは “\” キーは “¥” キーのことである。

13: 行目の (p(i) * 2) Mod 10000 は、繰り上がらない部分の処理だ。除算に \ を使う場合と Mod を使う場合で、計算結果がどうなるか理解できるだろうか。If 文を使っても同様の効果を得ることはできるが、わざわざ手の込んだ式を使っている。ところで 12: 行目と 13: 行目の処理順を逆にすると正しい結果にならない。なぜかは、ちょっとした頭の体操程度に考えてもらいたい。

さて、これで 2^{100} の計算はできた。次はこれを表示しなくてはならない。表示は上の桁から順に並べればよい。そのために 3: 行目で表示の仕方を指定したのだ。この指定がないと例えば p(9) = 1234、p(8) = 987 のときは、本来 12340987... となるべきものが 1234987... となってしまうからだ。"0000" の書式指定は 4 桁を保証するために必要なのだ。これでめでたく 12340987... のように表示されるわけである。

もっと計算桁数を増やしたければ、配列変数を増やせばよい。**VBA** では Long 型が 4 バイトだから p(i) に 8 桁の数を与えられる。10 個つなげて 80 桁だ。それなら 2^{200} の計算も軽くできる。

最後に、p(10) がどうなったかに触れておこう。11: 行目の For 文により最初は i = 9 で始まる。するとプログラムは 12: 行目で p(10) = p(10) + (p(9) * 2) \ 10000 の代入をすることになる。もし配列変数が p(9) までしか用意されていないと、エラーを返すことがある。だからダミーの配列変数 p(10) が必要だったのだ。