6.2 完全数を探す

マクロ [DispOfCM.vba] で約数をすべて列挙できたので、それらの和をとれば完全数を探すのに役立つだろう。和を求めるために変数 sum を追加しておこう。

programming list [PNsearch.vba]

```
 1 : Sub PNsearch()
 2 : Dim n, i, col, sum As Integer
 3 :
 4 :     n = Sheet1.Cells(1, 1).Value
 5 :     col = 1: Sheet1.Cells(2, col) = 1
 6 :
 7 :     sum = 1
 8 :     For i = 2 To n / 2
 9 :         If n Mod i = 0 Then
10 :             col = col + 1: Sheet1.Cells(2, col) = i
11 :             sum = sum + i
12 :         End If
13 :     Next i
14 :
15 :     Sheet1.Cells(3, 1) = sum
16 : End Sub
```

今度のマクロは、自分自身を含まない約数の和を表示させるのが目的である。その際、自分自身以外の約数をワークシートの 2 行目に、その和を A3 セルに表示することにした。例えば 8 に対しては、2 行目に 1, 2, 4 までが表示されその下に 7 が表示される。ちなみに 8 は完全数ではないので、和が 8 になっていないことも指摘しておこう。

プログラムは sum に関わる部分だけが変更されている。7: 行目に追加したのは sum = 1 である。この時点でセルに 1 を表示したので、同時に sum に 1 を代入したのだ。

同様に 11: 行目は、約数をひとつ表示するたび、それを sum に加えている。

15: 行目で和を表示したら終了だ。自分自身は和に加えないので n を表示させていない。ここでは sum の値を表示させれば十分だ。

では、このマクロで 6 と 28 が完全数であることを確認してほしい。また、いくつかの数をセルに入力して完全数を探してもらいたい。運良く見つかれば大したものである。おそらく [PNsearch.vba] では完全数を見つけられなかったのではないか？　それもそのはず、100 や 200 までには完全数は存在しないのだ。そうなると行き当たりばったりの入力では発見は難しいだろう。そこで、和が入力した数と等しくなったとき、その数を出力するようにすればよい。そうすれば、コンピュータは次の完全数を見つけるまで計算をしてくれる。君たちはお茶でも飲みながら、気楽に待っていればいいのだ。もっとも、コンピュータはお茶を入れる時間を与えてくれないだろうが。

programming list [PNfind.vba]

```
 1 : Sub PNfind()
 2 : Const SUP As Integer = 1000
 3 : Dim n, i, row, sum As Integer
 4 :
 5 :     row = 1
 6 :     For n = 6 To SUP
 7 :         sum = 1
 8 :         For i = 2 To n / 2
 9 :             If n Mod i = 0 Then
10 :                 sum = sum + i
11 :             End If
12 :         Next i
13 :
14 :         If n = sum Then
15 :             row = row + 1: Sheet1.Cells(row, 1) = n
16 :         End If
17 :     Next n
18 : End Sub
```

マクロは [PNsearch.vba] に手を加えたものである。ワークシートの値を読み取らないので、それに関わる文は不要になった。

その代わり上限 SUP までにある完全数を見つけるため、2: 行目で SUP を設定し、6: 行目から For-Next 構文を用いて探すことにしてある。最初の完全数は 6 であることが分かっているので、n = 6 から始めている。

8: 〜 12: 行目は [PNsearch.vba] と同様だが、いちいち約数を表示する手間を省いている。そのためセルに代入する文はない。

14: 行目が、完全数かどうかの判断をするところだ。完全数であれば 15: 行目でそのことが知らされる。実際にマクロを実行して 3 番目の完全数を探してもらいたい。さらに、SUP を 10000 にして、4 番目の完全数を探すとどうなるか？

やってみると容易に分かるように、完全数は非常に少ない。5 番目の完全数を探すのはさらに骨が折れるだろう。それもそのはずで、完全数は $2^{n-1}(2^n-1)$ の形をしている。今探した 4 つの数でちょっと確認してほしい。これでは指数関数的に大きな数になってしまうから、完全数がどんなにたくさんあったとしても、気軽に発見できないのだ。

そこで忠告をしておこう。このマクロで 5 番目の完全数を探す暴挙に出ないように。5 番目の完全数は 8 桁の数だ。これは Long 型の守備範囲だと考えないでほしい。ここでのアルゴリズムは、8128 を見つけるのでさえ、少々時間を要する。単純に考えても、1 桁増えるごとに計算時間は 10 倍になるから、8 桁の解を見つけるには 10000 倍の時間がかかる。8128 を 0.1 秒で見つけられても、5 番目の完全数を見つけるには 1000 秒 (16 分 40 秒) もかかるのだ。しかし悪いことばかり

ではない。今度は、お茶を入れる時間が十分与えられるからね。

万が一にも、このマクロで 5 番目の完全数を探す誘惑にかられたら、どのぐらいの時間がかかるか予測すべきだ。例えば n の値によって、Select-Case 構文で分岐させるのもひとつの手だ(Select-Case 構文で調べても問題の解決にならないので、悪いが Select-Case 構文については各自で調べてもらいたい)。Case 10000, Case 100000, Case 1000000 のたびに n の値を表示させれば、8 桁の数にたどり着く時間の目安が立つ。すると、このマクロが役立たずであることが発覚するはずだ。そして、途中でマクロを中断したいと思うことだろう。そういうときのために、プログラム中に DoEvents 命令をひっそりと入れるのは効果がある。試してみるとよいだろう。そして結局 $2^{n-1}(2^n-1)$ を利用するのが得策だと気付くだろう。これを利用すれば、5 番目の完全数はすぐ見つかる。

結局のところ、コンピュータの計算速度を過信してはいけないということだ。そのために、どうしても人の手で、効率的なアルゴリズムが必要になるのである。手っ取り早く 5 番目までの完全数を知りたいなら、$2^{n-1}(2^n-1)$ の形の数だけ調査するプログラムに変更すればよい。驚くほど早く結果を目にすることができる。