6.2 完全数を探す

プログラム [dispofcm.cpp] 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.cpp]

```
1: #include <iostream>
2:
3: int main() {
4:     int n, i, sum;
5:     while(1) {
6:         std::cout << "input your candidate : "; std::cin >> n;
7:         std::cout << n << " :: 1 "; sum = 1;
8:         for(i = 2; i <= n/2; i++) {
9:             if(n % i == 0) {
10:                 std::cout << "+ " << i << " "; sum += i;
11:             }
12:         }
13:         std::cout << "= " << sum << std::endl;
14:     }
15:
16:     return 0;
17: }
```

今度のプログラムは、自分自身を含まない約数の和を “$8 :: 1 + 2 + 4 = 7$” のように表示させるのが目的である。そのため各 std::cout は、目的に添うように少々手を加えてある。:= が :: に変わっているのもその一環だ。それぞれの std::cout を注意深く見てもらいたい。ところでこの例では、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。

まず 4:行目に sum を追加した。sum には求めた約数の和が次々と加えられていく。

7:行目に追加したのは sum = 1; である。この時点で画面には “** :: 1 ” が表示されるので、同時に sum に 1 を代入したのだ。画面表示と和の計算が同時進行で行われる意味も兼ねて、1 行に 2 つの文を書いてみた。

同様に 10: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。ここでも 1 行に 2 つの文を書いたが、sum += i; の表現が目新しい。これは sum = sum + i; を省略して書いたものだ。何も省略した書き方をしなくてもよいけれど、こういう書き方にも慣れておくほうが得策だ。**C++**のプログラムを見ると、この書き方をよく目にするはずだ。それにこの書き方は、以前 i += 2; で目にしている。意味は i が 2 ずつ増えることだったと思うが、実は i = i + 2; の省略形だったのである。

13: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 sum を表示させれば十分だ。

TRY! 6 と 28 が完全数であることを確認せよ。また、いくつかの数を入力して完全数を探してみよ。運良く見つかれば大したものだ。

おそらく [pnsearch.cpp] 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。私たちはお茶でも飲みながら、気楽に待っていればよいのだ。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.cpp]

```
 1: #include <iostream>
 2: #define SUP 1000
 3:
 4: int main() {
 5:     int n, i, sum;
 6:     for(n = 6; n <= SUP; n++) {
 7:         sum = 1;
 8:         for(i = 2; i <= n/2; i++) {
 9:             if(n % i == 0) {
10:                 sum += i;
11:             }
12:         }
13:         if(n == sum) {
14:             std::cout << n << " is a perfect number." << std::endl;
15:         }
16:     }
17:
18:     return 0;
19: }
```

プログラムは [pnsearch.cpp] に手を加えたものである。

5:行目で用意した変数 n, i, sum は [pnsearch.cpp] と同じだ。しかし、今回は繰り返し入力を要請しないので、while(1) 文と以下に続く std::cin 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 を定義し、6:行目から for 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。

7:–10:行目は [pnsearch.cpp] 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 std::cout はない。

13: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 14:行目でそのことが知らされる。

TRY! 3 番目の完全数を探してみよ。さらに、`SUP` を `10000` にして、4 番目の完全数を探してみよ。

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。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。このプログラムで 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。`long` 型なら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 10,000 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 1,000 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。

EX. 実際はお茶を入れる時間どころか、茶摘みの時期まで待たされるかもしれない。`for` 文が二重になっているからだ。外側の `for` で桁が増え、内側の `for` で計算量が同じだけ増える。単純に考えて、8 桁の解を見つける時間は 4 桁の解を見つける時間の何倍だろうか。それは、4 桁の解を見つけるのが 0.1 秒とした場合、何分？何時間？だろうか。

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。

EX. 4 番目までの完全数が $2^{n-1}(2^n-1)$ の形になっていることを確認せよ。

TRY! このままのプログラムでは、約数の表示がない。約数の和が表示できるよう、プログラムに修正を加えよ。