1.2 10 進数と 256 進数

[fibonacc.cpp] で試して分かったと思うが、フィボナッチ数列は結構大きな数になってしまう。君たちのコンピュータでは、何項目まで求めることができただろうか。コンピュータのことだから、無限に大きな値を扱うわけにはいかないが、やりようによっては数万桁の数でも扱える。しかし、この話題はしばらく後になる。

フィボナッチ数の計算に使った変数は `int` 型の整数だったはずだ。コンピュータが扱える数値に限界があるのは、変数の型による制約が大きい。私たちが日頃使っている数は 10 進数である。これは 1 桁あたり 10 種類の数を使える。携帯電話は大抵 090-xxxx-xxxx や 080-xxxx-xxxx の番号を持っているので、自由になる桁は 8 桁ある。各桁に 10 種類の数が使えるから、電話番号は

$$10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 10^8 = 1\text{ 億}$$

まで OK だ。090 だけでは残念ながら国民 1 人 1 人には行き渡らない計算だが、080 も使えば国民 1 人 1 人に確実に行き渡る。しかし、これらの番号は単に（通話のための）電話番号だけに使われる訳ではなく、（通話をしない）通信にも使われるので十分な数量を確保したことにはなっていないようだ。

ところで、**C++**の “桁” にあたる単位は “バイト” という。1 バイトは xxxxxxxx の 2 進数で処理されている。具体的には 11010010 や 01011011 などという種類の “数字” が使えるのだ。すると 1 バイトで $2^8 = 256$ 種類の “数字” になる。従って、**C++**は 256 進数ということなのだ。何だか目が回りそうになってないかい？　気を確かに持とう。

そこでようやく `int` 型の整数の話に入れる。`int` 型の整数は大抵 2 バイトか 4 バイトである。2 バイトなら $256^2 = 65{,}536$ 通りの数が使え、4 バイトなら $256^4 = 4{,}294{,}967{,}296$ 通りの数が使える。しかし `int` 型は整数を前提としている—つまり正の数と負の数を扱う—ので、2 バイトなら -32767〜32767 の範囲の整数が、4 バイトなら -2147483647〜2147483647 の範囲の整数が扱えるのだ（ビミョーに間違った言い方なので少し後で訂正しよう）。

0.2 節の TRY!でこれらの数を入力したことを覚えているだろうか。そのとき、妙な振る舞いをしたのなら、今その理由がおぼろげながら理解できるかもしれない。単純に考えれば 2 バイトの 2 進は 11111111 が最大数と思える。それが 10 進数の 32767 だとしたら、32768 は表せない。でも、コンピュータは何かを表示したはずだ。それは現時点の想像と一致するだろうか。謎の解明はもう少し待ってもらおう。

さて、大きな数を扱いたければ桁数を増やせばよいのだから、バイト値を増やせばいくらでも大きな桁を扱えることになる。もっとも、無制限に大きな桁を扱えるほど、型が用意されているわけではないのだけれど。`int` 型より大きい桁を扱いたいときは `long` 型を使えばよい。特にフィボ

ナッチ数は正の数しか扱わないので、unsigned long 型にすればよい。

programming list [fibonac2.cpp]

```
#include <iostream>
#define SUP 40

int main() {
    int i;
    unsigned long f1 = 1, f0 = 1;
    for(i = 3; i <= SUP; i++) {
        std::cout << f1 + f0 << " ";
        f1 = f1 + f0;
        f0 = f1 - f0;
    }
    std::cout << std::endl;

    return 0;
}
```

[fibonacc.cpp] に較べて変更点は、f1, f0 を unsigned long 型にしたことだけである。

TRY! #define SUP 40 の値を変更して、もう少し先のフィボナッチ数を求めてみよ。君たちのコンピュータは、何番目のフィボナッチ数まで計算できるだろうか？

TRY! scanf 文を用いて、整数 n を入力すると n 番目のフィボナッチ数だけを表示するよう、プログラムを変更せよ。

今、無造作に unsigned を使ってみた。実際に n 番目のフィボナッチ数を計算させれば分かることだが、unsigned がなくても、求められる最大のフィボナッチ数に大差はない。それはそうだ。2 バイトなら、扱える最大の正の数が 32767 から 65535 に増えるだけなのだから。

これで分かったと思うが、扱える桁数を増やすという理由で unsigned を使う意味はない。いや、unsigned はそういう理由だけで使ってはいけない。2 倍大きな数を扱うために unsigned を使うと、高い代償を払う羽目になるからだ。その一端は、後述するコラッツの問題で触れよう。