1.3 フィボナッチ数列の 2 項間の比

フィボナッチ数列には面白い性質がある。そのひとつは、隣り合う 2 項の比を調べることで見えてくる。1, 1 から始まるフィボナッチ数列を 2 項ずつペアにし、

$$\frac{1}{1}, \frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \frac{34}{21}, \cdots$$

のように分数を作る。この程度は暗算や電卓で計算できるだろう。

EX. 実際に比を計算してみよ。

しかし、こういったことを調べるには、やはりコンピュータが適している。比の計算ということは、分数、つまり割り算をしなくてはならない。割り算をすれば、必ずといってよいほど小数の値がでるものだ。しかし、割り算には特有の問題が含まれている。始めにそれを説明しよう。

programming list [ffailure.cpp]

```
 1: #include <iostream>
 2: #define SUP 40
 3:
 4: int main() {
 5:     int i;
 6:     unsigned long f1 = 1, f0 = 1;
 7:     for(i = 3; i <= SUP; i++) {
 8:         std::cout << (f1 + f0) / f1 << " ";
 9:         f1 = f1 + f0;
10:         f0 = f1 - f0;
11:     }
12:     std::cout << std::endl;
13:
14:     return 0;
15: }
```

プログラムは [fibonac2.cpp] の 8:行目を変えただけである。新しいフィボナッチ数を現在の f1 で割れば、2 項の比になる。

実行すれば分かることだが、電卓で計算したようにならなくてガッカリするだろう。この原因は 8:行目だが、遠因は 6:行目に存在している。8:行目の (f1 + f0) / f1 の計算は、6:行目より整数同士の割り算になる。まあ、これは当然だ。ところでコンピュータは、整数同士の演算は結果も整数にしてしまうし、浮動小数点数同士の演算は結果も浮動小数点数になるものと思っている。コンピュータにとっては 8/5 の演算結果は 1 なのだ。ところがコンピュータはちゃっかりしている。8.0/5.0 の演算なら 1.600000 と表示するのだ。

この性格を見越せば解決は簡単だ。浮動小数点数の答えがほしければ、浮動小数点数同士の割り算にしてやればよいだけだ。f1 と f0 は float 型にすればよい。だが、ついでだから double 型に

しておこう。これは**倍精度**浮動小数点型（double-precision floating type）と呼ばれ、`float` 型より精度がよい。

programming list [fratio.cpp]

```
1: #include <iostream>
2: #define SUP 40
3:
4: int main() {
5:     int i;
6:     double f1 = 1.0, f0 = 1.0;
7:     for(i = 3; i <= SUP; i++) {
8:         std::cout << (f1 + f0) / f1 << std::endl;
9:         f1 = f1 + f0;
10:        f0 = f1 - f0;
11:    }
12:
13:    return 0;
14: }
```

`f1` と `f2` の型を `double` にした際、`f1 = 1.0` のように小数点を付けて代入していることに注意を払ってほしい。6:行目の宣言が `double` であるから、実際は `1` として代入しても 8:行目の計算は浮動小数点数で行ってくれる。しかし、小数点があるほうが見た目に分かりやすい。

プログラムを実行すると、フィボナッチ数列の隣り合う 2 項の比が一定になっていく様子が分かるだろう。改行しながら表示したのは、小数値を並べて表示するのが見苦しいと思ったからである。そのため、ここでは `for` 文を抜けたところで改行をしていない。もっとも、ここの改行を残していても、1 行余分に改行が入るだけなので何の問題もないけれど。ところで、表示された値が何かは後の話題となる。

TRY! フィボナッチ数列は `f1 = 1.0`、`f0 = 1.0` で始めているが、これを違う値で始めると 2 項の比に変化があるか確かめよ。

TRY! 隣り合う 2 項の比でなく、ひとつおきの 2 項の比を計算すると、比の値はどうなるか？

今回、精度の向上を狙って `float` 型でなく `double` 型を選んだ。精度が向上することは良いことだが、余計にメモリを食うことも事実だ。どのみちコンピュータでは、無制限に高い精度を得られないのだから、`double` の使い過ぎは無駄遣いに通じる。ここでは `float` でも十分である。ちなみに、