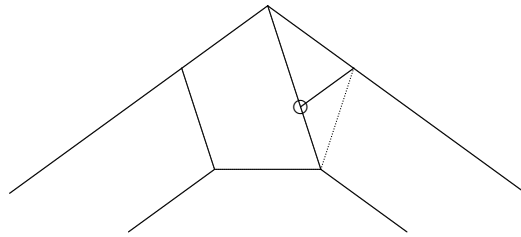


5.5 ここにも黄金比

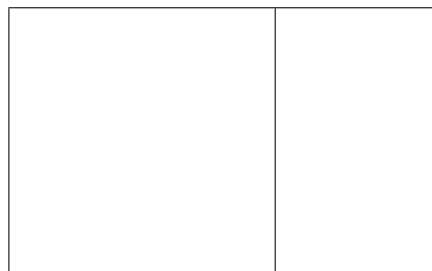
旅の途中に少し寄り道をしよう。

正5角形は意外と簡単に作れる。それには、適当な長さの紙テープを用意すること。紙テープをくると結んで、折り目をきちんとつけると正5角形ができる。



そして、ここにも黄金比が隠れている。図中、○印の位置で対角線が黄金分割されているのである。

また長方形には、美しい長方形と美しくない長方形がある。美しい長方形とは、長方形から正方形を切り取ったとき、残りの長方形がもとの長方形と相似になるものを言う。



なぜ美しいかと言えば、残った長方形から同じように正方形を切り取っても、次に残る長方形も再び相似形になるのだ。この操作は寸分たがわず繰り返すことができる。そして、この性質を持つ長方形は、縦・横の比が黄金比なのである。

紙の話題が続いたところで、現在私たちが使っている用紙にも触れておこう。これは、黄金比を持つ長方形より若干横の長さが短い。その比は $1:\sqrt{2}$ である。しかし、これも均整がとれた長方形だ。なぜなら、この長方形は2つ折りにしたときに、初めの長方形と相似形になる。すると、いくら半分に折っても形が変わらない利点がある。だから、定型用紙として普及しているわけだ。

2つ折りにすることは、用紙の面積が $1/2$ になることである。面積が $1/2$ になるということは、各辺の長さが $1/\sqrt{2}$ になることだ。だから、 $A3 \rightarrow A4$ の拡大率が $1/\sqrt{2} \approx 0.707$ になっているわけだ。

それでは、用紙の大きさは何が基準になっているか知ってるかい？ 基準になっているのは A0 と呼ばれる用紙だ。これを 2 つ折りにして A1、また 2 つ折りにして A2、またまた折って A3、... という具合だ。では A0 のサイズは？ 実は A0 は、面積が $1(\text{m}^2)$ である。基準の用紙らしい大きさだね。

EX. A0 の用紙は、縦・横比が $1 : \sqrt{2}$ で面積が $1(\text{m}^2)$ である。縦・横の長さはそれぞれ何 cm か。

用紙には A 判の他に B 判がある。B 判の基準は B0 と呼ばれる用紙で、これも基準の面積を持っている。用紙の縦・横比が $1 : \sqrt{2}$ だから、基準の面積も $\sqrt{2}(\text{m}^2)$ とすれば整合性があったのに、惜しいかな、 $1.5(\text{m}^2)$ の面積となっている。そのせいで、用紙は $B0 \rightarrow A0 \rightarrow B1 \rightarrow A1 \rightarrow B2 \rightarrow \dots$ の順に小さくなるが、拡大率は 0.816 と 0.866 が交互に出現するのだ。もし B0 の面積が $\sqrt{2}(\text{m}^2)$ であれば、拡大率は常に 0.841 だ。もっとも、B 判の用紙を使わなければ何も面倒なことはないけれどね。

ただ、A 判と B 判の両方を使う機会が多い人は、拡大率の対応表でも作っておくと便利だろう。何も C++ で表を作ることもないが、配列を利用して一覧表を仕上げてみるもの一興だろう。

programming list [octavo.cpp]

```

1: #include <iostream>
2:
3: int main() {
4:     int i, j;
5:     float t[2][6] = { {1.000, 0.707, 0.500, 1.224, 0.866, 0.612},
6:                       {0.816, 0.577, 0.408, 1.000, 0.707, 0.500} };
7:     for(i = 0; i <= 1; i++) {
8:         for(j = 0; j <= 5; j++) {
9:             printf("%.3f  ", t[i][j]);
10:        }
11:        std::cout << std::endl;
12:    }
13:    return 0;
14: }
```

このような表は、いわゆる行列の考えに基づいて作られる。[octavo.cpp] は

	A_n	A_{n+1}	A_{n+2}	B_n	B_{n+1}	B_{n+2}
A_n	1.000	0.707	0.500	1.224	0.866	0.612
B_n	0.816	0.577	0.408	1.000	0.707	0.500

の表の、数値部分だけを出力するものである。数値部分は 2 行 6 列の行列だ。

C++ で 2 行 6 列の行列を実現するには配列を使うとよい。この場合の配列は `t[2][6]` のように、`[]` を並べて記述することになっている。プログラムでは 5:-6: 行目にかけて、浮動小数点数を扱

う配列変数 `t[2][6]` を用意し、同時に右辺の `{}` で初期化している。初期化は行ごとに `{}` でグループ化し、それらをまとめて外側の `{}` で囲んでおく。コードは 2 行に分けて書く必要はないが、表の出来上がりをイメージするためにも 2 行に分けて書いていただけた。

表の出力は 7:–12:行目でなされる。ここでは 2 つの `for` 文が入れ子になっている。

`for` 文は外側の `i = 0` から始まるが、先に繰り返しをするのは内側の `for` 文である。すなわち `i = 0` のまま、`j` の値が 0 から 5 まで変化する。それに従って 9:行目の `printf` 文が実行されるので、順に `t[0][0]`, `t[0][1]`, `t[0][2]`, ..., `t[0][5]` が出力される。配列は 6 列だが、列番号は 0 から 5 に対応していることに注意してほしい。

`printf` 文は「べき乗の計算」の節で登場している。このときは、書式 `%04d` は整数値を空位に 0 を入れて 4 桁表示するのだと言った。ここでは、書式に `%.3f` を使っている。中央の `.3` の意味は、小数点以下 3 桁を表示するということだ。そうした理由は、`%.3f` では一般に浮動小数点数を小数点以下 6 桁まで表示するからだ。それだと、ちょっとうざったい。ちなみに `%d` や `%f` のことは、「循環小数の秘密」の節で早々に話したけれど、もう忘れちゃってるよね。

ひと通り `j` に関する `for` 文が出力されたら、プログラムは次の `i` へ移る。入れ子になった `for` 文は常にこのパターンで繰り返しが行われるのだ。

ただし、ここでは表に仕立てたいので、一旦 `j` に関する `for` 文を抜けたら改行しておかなくては行けない。それが 11:行目である。

TRY! `[octavo.cpp]` では項目行などが表示されない。項目行なども表示するよう、プログラムを修正せよ。

おっと、いかん。寄り道が過ぎたようだ。迷子になる前にもとの道へ戻るとしよう。