9.2 マンデルブロ集合

[infcomp.cpp] で複素数の発散が見えるようになったものの、漸化式の計算を繰り返すのは億劫（おっくう）である。そして、その必要もない。

複素数 $((r,\ \theta))$ は 2 乗すると $((r^2,\ 2\theta))$ になることは前に述べた。その際、複素数の大きさを決めるのは r であるから、r が 1 を超えたらもう発散するしかない。だったら、r の大きさで発散を判断してもいいのかな？　残念ながらそれはまずい。漸化式は $z_{n+1} = z_n^2 + c$ なので、r が 1 を超えても c を加えることで $r < 1$ になるかもしれない。しかし [infcomp.cpp] の挙動から、発散するときは 20 回の計算よりだいぶ前に発散していたように思える。そこで、20 回程度の計算で収束・発散を判断してもかまわないだろう。

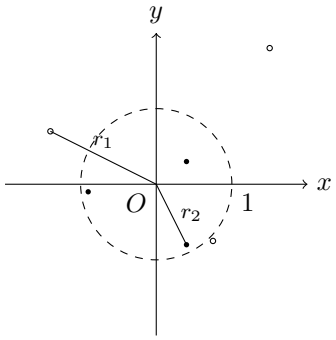
programming list [infcomp2.cpp]

```
1: #include <iostream>
2: #include <complex>
3:
4: void recur(std::complex<double> c) {
5:     std::complex<double> z(0, 0);
6:
7:     for(int i = 0; i < 20; i++) {
8:         z = z * z + c;
9:     }
10:    if(abs(z) > 2) {
11:        std::cout << "inf (divergence!)" << std::endl;
12:    } else {
13:        std::cout << "nan (convergence!)" << std::endl;
14:    }
15: }
16:
17: int main() {
18:    std::complex<double> c;
19:
20:    std::cout << "input '(a, b)' as a+bi: ";
21:    std::cin >> c;
22:
23:    recur(c);
24:
25:    return 0;
26: }
```

10:行目から、複素数の絶対値は abs(z) で求めていることが分かる。数学感覚に近くていいね。収束・発散の判断はこれで十分だ。大きさが 2 を超えたらまず発散するだろうから。もし不安があれば、繰り返しの回数を増やすか、もっと大きな値（たとえば > 10 など）で判定すればよい。

さて、これが何の役に立つのだろうか。君たちは**マンデルブロ**[1]**集合**という名前を聞いたことがあるだろうか。

いま扱っている複素数の漸化式は $z_{n+1} = z_n^2 + c$ である。前節で見たように、漸化式は c の値によって収束したり発散したりする。つまり c の値には、収束するグループと発散するグループを分ける役目があるのだ。直観では、原点から遠い複素数ほど発散しやすいと思える。実際そうであるが、では、収束する複素数と発散する複素数の境目はどこだろうか。



x-y 実数平面上の点は複素平面のようにいかないが、原点からの距離だけに注目すれば、2 乗するたびに距離が縮まるのは半径 1 の円内にある点である。円外の点は発散する。複素数も 2 個の組なので、まあ円に近い形が収束・発散の境目になる ... と思ったら大間違いである。

programming list [mandel.cpp]

```
#include <iostream>
#include <complex>

int recur(std::complex<double> c) {
    std::complex<double> z(0, 0);

    for(int i = 0; i < 20; i++) {
        z = z * z + c;
    }
    if(abs(z) > 2) {
        return 1; // divergence.
    } else {
        return 0; // convergence.
    }
}

int main() {
    for(int y = 0; y < 40; y++) {
        double im = (20 - y) / 16.;

        for(int x = 0; x < 40; x++) {
```

[1] プノワ・マンデルブロ (1924–2010)：フランスの数学者・経済学者。

```
22:             double re = (x - 32) / 16.;
23:
24:             std::complex<double> z(re, im);
25:             if(recur(z) == 1) {
26:                 std::cout << " 1";
27:             } else {
28:                 std::cout << "  ";
29:             }
30:         }
31:         std::cout << std::endl;
32:     }
33:     std::cout << std::endl;
34:
35:     return 0;
36: }
```

マンデルブロ集合を Terminal やコマンドプロンプトで見るのは狂気の沙汰だが、このプログラムで何とかそれっぽいものが見られる。もしかして君たちは、もっと細かく描画された、場合によっては色分けされた図をインターネットで見たことがあるかもしれない。そう、この場合の収束・発散の境目は実はとんでもなく複雑なのだ。

`recur()` 関数は発散（divergence）・収束（convergence）を表示するのではなく、発散なら `1` を、収束なら `0` を返す。この関数は 25:行目で判定に使われ、発散なら画面に "␣1" を、収束なら画面に "␣␣" を表示する（"␣" は空白 1 個の意味）。18:行目と 21:行目の `for` 構文から、$x \times y$ を 40×40 に想定していることが見て取れるだろうが、Terminal の画面は横方向の文字間隔より縦方向の改行間隔の方が広い。つまり、横方向に律儀に 1 文字づつ出力すると、横が詰まって縦に間延びして見えてしまう。それを少しでも和らげるため、横方向は一度に 2 文字分出力したのだ。よって、ウィンドウサイズは横 80 文字が入るように、あらかじめ調整しておいてほしい。

しかし、x, y の値をそのまま使ったのでは、x-y 座標を模していることにならない。そこで 19:行目と 22:行目で、$-2 \leq x \leq 0.5$、$-1.25 \leq y \leq 1.25$（正方形の範囲）になるように変換している。この範囲に変換したのはもちろん、マンデルブロ集合がおおむね収まる範囲だからである（x 軸は -2.5 程度まで延びていると全部収まる）。

19:行目と 22:行目では `y` と `x` で引き算の仕方が逆であることを注意しておこう。このコードでは、`y`, `x` ともに `0` から `40` へ変化させているので、この引き算の仕方で $y:\ 1.25 \to -1.25$、$x:\ -2 \to 0.5$ と変化する。当然、画面の出力は上から順に行われるので、とくに `y` は値が大きい方から表示させなくてはならない。引き算の仕方を同じにしたければ、`for` 構文の増減を逆にすればよい。ま、好みの問題だな。