10.2 パラパラ画の準備

2 次元配列を用いて、なんとか格子盤面に石を配置することができた。ただ、これを Terminal の画面で表示すると縦長になっていたはずだ。マンデルブロ集合で指摘したように、画面の横幅が詰まっているせいだ。マンデルブロ集合では横幅を広くするため、空白 “␣” を 1 個入れたのだが、この場合は隙間を開けるのは好ましくない。そこで、2 **バイト文字**を表示させれば一文字でも横幅は広がるので都合がよいだろう。

Terminal やコマンドプロンプトで使用する文字の設定がどうなっているか調べて、できることなら**等幅（とうはば）**のフォントで表示させよう。この例では、石を “∗” で、空きマスを “・” で表すことにした。クラス部分だけ示しておこう。

programming list [Life2nd.cpp]

```
class Lifegame { // クラス部分
    int p[6][6];
    std::string d[6][6];

public:
    Lifegame(int s[][6], int rows) {
        for(int x = 0; x < rows; x++) {
            for(int y = 0; y < 6; y++) {
                p[x][y] = s[x][y];
                if(s[x][y] == 1) {
                    d[x][y] = "∗";
                } else {
                    d[x][y] = "・";
                }
            }
        }
    }

    void disp() {
        for(int x = 1; x < 5; x++) {
            for(int y = 1; y < 5; y++) {
                std::cout << d[x][y];
            }
            std::cout << std::endl;
        }
        std::cout << std::endl;
    }
};
```

クラスは、複素数を見て回ったとき一度目にしていたと思う。当時とは `public:` の位置が違うことに気づいたかな。数学の旅なので **C++** を詳しく説明しないが、この地ではクラスに含まれるメンバ関数だけを `public:` 指定した。では、`public:` 指定から外れた配列変数はどうなったかという

と、それは明示されていないが private: になった。private: なものは外部から参照できない。つまり p[][], d[][] はメンバ関数から参照できても、main() からは参照できない仕様にしたのだ。この方が安全であるというのが **C++**の主張らしい。もっとも自分専用のクラスなんだら、全部 public: でもかまわないんだけどね。でも、これで p[][], d[][] に格納する値は安全が保証されたのだ。

と言っても、p[][] に文字を代入するのではまずい。それは p[][] が int 型であるという理由ではなく、実は p[][] は石の 活き/排除/誕生 を調べるために整数値を用いたのであった。だから、画面表示用には別の配列を用意する必要がある。したがって、初期配列 s[][] の値を p[][] に代入すると同時に、s[][] の値によって表示させる文字を配列 d[][] に代入している。d は string 型である。代入は 10:–14:行目で行われる。当然、表示するのは p[][] ではなく d[][] なのだから、22:行目は d[x][y] を std::cout へ送っている。

では、石の 活き/排除/誕生 を判定するメンバ関数を追加しよう。

programming list [Life3rd.cpp]

```
 1: #include <iostream>
 2:
 3: class Lifegame {
 4:     int p[6][6];
 5:     int t[6][6];
 6:     std::string d[6][6];
 7:
 8: public:
 9:     Lifegame(int s[][6], int rows) {
10:         for(int x = 0; x < rows; x++) {
11:             for(int y = 0; y < 6; y++) {
12:                 p[x][y] = s[x][y];
13:                 if(s[x][y] == 1) {
14:                     d[x][y] = "＊";
15:                 } else {
16:                     d[x][y] = "・";
17:                 }
18:             }
19:         }
20:     }
21:
22:     void chkaround() {
23:         for(int x = 1; x < 5; x++) {
24:             for(int y = 1; y < 5; y++) {
25:                 t[x][y] = p[x-1][y-1] + p[x-1][y  ] + p[x-1][y+1]
                            + p[x  ][y-1]               + p[x  ][y+1]
                            + p[x+1][y-1] + p[x+1][y  ] + p[x+1][y+1];
26:             }
27:         }
28:     }
```

```
29:
30:     void disp() {
31:         for(int x = 1; x < 5; x++) {
32:             for(int y = 1; y < 5; y++) {
33:                 std::cout << t[x][y];
34:             }
35:             std::cout << std::endl;
36:         }
37:         std::cout << std::endl;
38:     }
39: };
40:
41: int main() {
42:     int s[][6] = {{0, 0, 0, 0, 0, 0},
43:                   {0, 0, 1, 0, 0, 0},
44:                   {0, 0, 0, 1, 0, 0},
45:                   {0, 1, 1, 1, 0, 0},
46:                   {0, 0, 0, 0, 0, 0},
47:                   {0, 0, 0, 0, 0, 0}};
48:     Lifegame cells = Lifegame(s, 6);
49:
50:     cells.chkaround();
51:     cells.disp();
52:
53:     return 0;
54: }
```

判定のためのメンバ関数 `chkaround()` は 22:–28:行目である。長ったらしい式だが、要するに `p[x][y]` の周り 8 マスの値を合計している。このために `p[][]` に格納する値は、石があるときは `1`、空きマスのときは `0` にしたのであった。合計値が石の個数を表す。

蛇足ながら 25:行目の式は、紙面に収まらないから改行して書いたのではなく、実際のコードでもこの形で入力している。**C++**は ; が文の区切りなので、体裁を整えるために改行して記述できるのはありがたい。

ただし、合計値は `p[x][y]` に代入するのではない。他の `p[][]` の計算をするときに値が変わっていたらまずいからだ。そのため、`p[x][y]` のマス周りの石の数を格納するために配列 `t[x][y]` を用意したのである。代入先が `t[][]` であることに注意してもらいたい。それに、石の置き換えは全部のマスに対して同時に行われなければならい。よって、`p[][]` の値は置き換え直前まで保たれている必要がある。

このプログラムはテスト用で、`main()` は各マスの周りに石がいくつあるかを表示するためのものである。だから、今回はマスの周りの石の数が格納されている `t[x][y]` を `std::cout` へ送ったのである。目まぐるしくてごめんね。もしここを `std::cout << d[x][y]` のままにすれば、“表の顔” を見ることになる。その裏で石の数を数えているのだ。あとはこれが連続して見られれば、

ノートの隅に少しずつ変化する絵を描いてパラパラ漫画にしたように、変化する表示を見られるだろう。

ところで気づいていると思うが、画面に表示する配列はクラスで定義した配列のすべてではない。表示しなかったのは、前節の図で “-” を打ったマスである。実はこのマスはダミーである。盤面の端について周りのマスの石の数を数えるとき、端のマスは内側のマスとは状況が異なる。もちろん、状況が異なれば相応のコードを書けばよいのだが、やはり簡単に済ませたい。そのために、盤面の端は石を置くためのマスではなく、計算をするためのマスにしたのである。