プログラミングおよび実習 I・第 7 回

2003 年 5 月 28 日

1　オセロゲームの局面の表現

オセロゲームのゲーム途中に現れる 1 つ 1 つの局面は

1. 8×8 のマスの状況 (黒石がある / 石がない / 白石がある) と
2. 手番 (次は黒の番 / 次は白の番)

の 2 つで決まるはずです。前回までに作成したプログラムでは、この 2 つを、関数 main の中で

```
int board[BOARD_SIZE][BOARD_SIZE];
int turn;
```

と定義された配列と変数の組で表現していました。オセロゲームの進行は、キーボードからのユーザーの入力に従って、この配列 board と変数 turn の組が変化していくことに他ならないわけですが、そう考えてみると、これまでのプログラムには若干不自然な点があることに気づきます。

たとえば、付録 (7 – 10 ページ) の othello16.c では、関数 main の冒頭で局面の初期化を行っているわけですが、まず、 156 行目の

```
int turn = BLACK;
```

で変数 turn を初期化 (156 行目) した後、 158 行目の

```
reset(board);
```

で関数 reset を呼び出して、配列 board の初期化をさせています (158 行目)。変数 turn の初期化と配列 board の初期化は、オセロゲーム局面 (turn と board の組) の初期化の一部なのですから、関数 reset で (あるいは関数 main 自身が) どちらもまとめて初期化する方が自然です。

また、このプログラムでは、プレーヤーが石を置いたときの配列 board の変化を関数 putDisk の中 (107 〜 111 行目)

```
for (dx = -1; dx <= 1; dx++)
    for (dy = -1; dy <= 1; dy++)
        if (dx != 0 || dy != 0)
            checkDir(b, t, x, y, dx, dy);
b[x][y] = t;
```

で処理し、それと同時に手番が変る処理は、関数 main の中 (163 行目)

```
turn = OPPONENT(turn);
```

で行っています。どちらもプレーヤーが石を置いたことによる局面 (turn と board の組) の変化ですから、本来ならこれも 1 個所にまとめるのが自然です。

このように、オセロゲームの局面を扱う関数は、盤面の状況を表す 8×8 の int 型の 2 次元配列型のデータと、次の手番を表す int 型のデータの 2 つを、 1 組のまとまりのあるデータとして処理するのが自然です。次節では、 C のプログラム中で、このようないくつかのデータの組をひとまとまりのデータとして取り扱う方法について解説します。

2　構造体

オセロゲームの局面のように、いくつかのデータの組として構成されるひとまとまりのデータを、C では、**構造体**と呼ばれるデータ型として扱うことができます。

2.1　構造体の宣言

構造体型は C の予約語 struct を使って、プログラム中でつぎのように宣言することにより利用可能になります[1]。

```
struct 構造体タグ {
    型指定1   メンバ名1;
    型指定2   メンバ名2;
    型指定3   メンバ名3;
      ⋮          ⋮
    型指定n   メンバ名n;
};
```

「構造体タグ」はここで定義する構造体に付ける名前 (識別子) で、変数名や関数名などと同じように適当に選びます[2]。構造体型のデータを構成している各部分を、その構造体の**メンバ**と呼びます。構造体の宣言の { と } で囲まれた部分で、この構造体のメンバのデータ型と、そのメンバを指し示すときに用いる名前 (メンバ名[3]) を指定します。この部分の書き方は、ちょうど変数や配列の宣言と同じ形になっていることに注意してください。

たとえば、オセロゲームのプログラムで、配列 board と変数 turn に記憶されていた 1 つの局面を表すデータ型は、つぎのような構造体として宣言できます。

```
struct OSTATE {
    int board[BOARD_SIZE][BOARD_SIZE];      /* 盤面の状況 */
    int turn;                               /* 次の手番 */
};
```

また、年月日の 3 つの情報で構成される日付を表すデータは型は、

```
struct DATE {
    int year;       /* 年 */
    int month;      /* 月 */
    int day;        /* 日 */
};
```

のように宣言される構造体として表現できるでしょう。

構造体の宣言はあくまでデータ型の宣言ですから、これで何らかの変数や配列が準備されるわけではないことに注意してください。構造体型のデータを記憶するためには (次節で紹介するように) その構造体型の変数 (や配列) を宣言 (定義) して、それを利用することになります。

---

[1]最後の ; を忘れないように注意しましょう。構造体の宣言は、関数定義の外やブロックの始まりなど、変数の宣言を行うことができる場所に書くことができます。構造体の宣言が有効となる範囲も変数の宣言の場合と同様です

[2]構造体タグは他の変数や関数の名前と重なっても構いません

[3]変数名や関数名などと同様な名前を適当に選んぶことができます。メンバ名についても他の変数や関数と同じ名前を使って構いません。また、異なる構造体が同じ名前のメンバを持っていても区別されます

2.2 構造体型の変数宣言

構造体型も 1 つのデータ型ですから[4]、その型のデータを変数に記憶することができます。構造体型のデータを記憶するための変数は、その変数のデータ型を「struct 構造体タグ」の形式で指定して宣言します。たとえば、前節のように OSTATE や DATE という構造体が宣言されている場合、

```
struct OSTATE s, t;
struct DATE today;
```

のように、これらの構造体型のデータを記憶する変数 t、s、today を宣言 (定義) することができます。これらはちょうど、int 型の変数 x、y の宣言

```
int x, y;
```

の int が、struct OSTATE や struct DATE に置き換わった形になっていることに注意してください。

変数の宣言は、構造体の宣言と一緒にして、

```
struct OSTATE {
    int board[BOARD_SIZE][BOARD_SIZE];      /* 盤面の状況 */
    int turn;                               /* 次の手番 */
} s, t;

struct DATE {
    int year;       /* 年 */
    int month;      /* 月 */
    int day;        /* 日 */
} today;
```

のようにすることもできます[5]。

2.3 構造体型の変数の初期化

構造体型の変数を宣言と同時に初期化するためには、次の例のように変数名に続いて、= と、各メンバの初期値を , で区切って { } の中に書きます[6]。

```
struct OSTATE s = { { { NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE },
                      { NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE },
                      { NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE },
                      { NONE, NONE, NONE, WHITE, BLACK, NONE, NONE, NONE },
                      { NONE, NONE, NONE, BLACK, WHITE, NONE, NONE, NONE },
                      { NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE },
                      { NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE },
                      { NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE } },
                    BLACK };
struct DATE today = { 2003, 5, 28 };
```

この {} を使った形式の値の設定は、変数の宣言時にしか使用することはできません。たとえば、

```
struct DATE today;
```

---

[4]当然、構造体型の要素からなる配列や、別の構造体型のメンバを持つ構造体型などのデータ型を使うことができます

[5]この場合、これらの構造体の定義を他の場所で使う予定がないのであれば、構造体タグの「OSTATE」や「DATE」は省略しても構いません

[6]配列を初期化する場合も {} の中に配列要素の初期値を並べて書きますから、2 次元配列をメンバとする構造体 OSTATE の初期化では、{} が 3 重の入れ子となっています

のように変数 today を宣言した後に、

```
today = { 2003, 5, 28 };
```

と書いて、この変数の各メンバに代入を行うことは許されませんので注意が必要です。次節に紹介する方法で、構造体のメンバそれぞれに対して個別に値を代入します。

2.4 構造体メンバへのアクセス

構造体の各メンバにアクセスするためには、**直接メンバ演算子** "." を用います。s がある構造体型のデータ (を表す式) で、m がその構造体のメンバ名のとき

$$s.m$$

という式は、s の m というメンバの値を意味します。s が変数や配列要素などの代入可能な対象[7]である場合は、この式を代入演算子 = の左辺に書いて、s の m というメンバだけを変更することもできます。たとえば、先のように宣言された変数 s、t があった場合、

```
FOR_EACH_POS(x, y)
    s.board[x][y] = t.board[x][y];
s.turn = OPPONENT(t.turn);
```

のようなプログラムを書くことが可能です。

2.5 構造体全体のコピー

次のプログラム tomorrow1.c のように、構造体型のデータ全体を変数に代入したり、関数の実引数として関数に渡したり、関数の返り値として呼び出し元に返したりすることも可能です。いずれの場合も、その構造体を構成しているすべてのデータ (メンバ) のコピーが起きます。

tomorrow1.c

```
 1 #include <stdio.h>
 2
 3 struct DATE {
 4     int year;
 5     int month;
 6     int day;
 7 };
 8
 9 #define IS_LEAP_YEAR(y) ((y)%4 == 0 && ((y)%100 != 0 || y %400 == 0))
10
11 /* 引数 d の次の日を返す */
12 struct DATE nextDay(struct DATE d)
13 {
14     /* 各月の日数 */
15     int dom[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
16
17     if (IS_LEAP_YEAR(d.year))                /* うるう年か? */
18         dom[1]++;
19     if (++d.day > dom[d.month - 1]) {        /* 次の月になるか? */
20         d.day = 1;
```

[7]代入演算子の左辺に書けるようなものを**左辺値**と呼びます。変数や配列の要素が左辺値の代表的な例です。左辺値である構造体の各メンバはやはり左辺値となります

```
21          if (++d.month > 12) {                   /* 次の年になるか? */
22              d.month = 1;
23              d.year++;
24          }
25      }
26      return d;
27 }
28
29 int main()
30 {
31      struct DATE today = { 2003, 5, 28 }, tomorrow;
32
33      tomorrow = nextDay(today);
34      printf ("今日は西暦%d年%d月%d日です\n",
35              today.year, today.month, today.day);
36      printf ("明日は西暦%d年%d月%d日です\n",
37              tomorrow.year, tomorrow.month, tomorrow.day);
38      return 0;
39 }
```

2.6 構造体へのポインタ

第 3 回では、配列全体を一度に代入したり、関数の引数や返り値として配列全体をやり取りすることができないことを説明しましたが、構造体のメンバの 1 つとして配列を含ませておけば、この構造体を利用することで (実質的に) 配列全体の代入や、関数間での配列全体のやり取りも可能となります。だだし、その構造体に含まれるすべてのデータのコピーが起りますので、構造体そのものをコピーしながら取り扱うのではなく、構造体へのポインタ値 (アドレス) を取り扱う方が効率的な場合が多くあります。これは、構造体のメンバに配列が含まれていない場合でも同様です。

構造体型へのポインタ (構造体型へのポインタ型の変数) は、他のデータ型へのポインタと同様に、つぎの例のように宣言することができます。

```
struct OSTATE *p;
struct DATE *q;
```

関数の仮引数となる場合も同様です。

p が構造体へのポインタ型の値 (アドレス) を表す式で、 m がその構造体のメンバ名のとき、

$$p\texttt{->}m$$

という式で p が指す構造体 (アドレス p に格納されている構造体) のメンバ m にアクセスすることができます。この "->" は**間接メンバ演算子**と呼ばれます。「p->m」という式は「(*p).m」と同じ意味となることに注意してください。先に紹介した tomorrow1.c というプログラムを、構造体へのポインタ値を引数として関数 nextDay を呼び出すように書き直すと、次のようなプログラムとすることができます[8]。

tomorrow2.c

```
1 #include <stdio.h>
2
3 struct DATE {
```

[8]書き換えた行に * 印が付してあります

```
  4     int year;
  5     int month;
  6     int day;
  7 };
  8
  9 #define IS_LEAP_YEAR(y) ((y)%4 == 0 && ((y)%100 != 0 || y %400 == 0))
 10
 11 /* 引数 d を次の日に進める */
*12 void nextDay(struct DATE *d)
 13 {
 14     /* 各月の日数 */
 15     int dom[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
 16
*17     if (IS_LEAP_YEAR(d->year))          /* うるう年か? */
 18         dom[1]++;
*19     if (++d->day > dom[d->month - 1]) { /* 次の月になるか? */
*20         d->day = 1;
*21         if (++d->month > 12) {          /* 次の年になるか? */
*22             d->month = 1;
*23             d->year++;
 24         }
 25     }
*26 }
 27
 28 int main()
 29 {
 30     struct DATE today = { 2003, 5, 28 }, tomorrow;
 31
*32     tomorrow = today;
*33     nextDay(&tomorrow);
 34     printf ("今日は西暦%d年%d月%d日です\n",
 35             today.year, today.month, today.day);
 36     printf ("明日は西暦%d年%d月%d日です\n",
 37             tomorrow.year, tomorrow.month, tomorrow.day);
 38     return 0;
 39 }
```

3 typedef による型定義

構造体を使ったプログラムに何度も「struct 構造体タグ」という型指定が現れて煩わしい場合は、C の予約語 typedef を使って、構造体型に名前をつけて置きましょう。typedef による型定義は

```
typedef 型指定 新しい型名;
```

のような書式で書きます。たとえば、

```
typedef struct OSTATE OState;
```

と書くと、これ以降 OState を OSTATE という構造体型を意味する型名として利用することができます。構造体の宣言と一緒にして、

```
typedef struct OSTATE {
    int board[BOARD_SIZE][BOARD_SIZE];
    int turn;
} OState;
```

のようにすることも可能です。構造体タグの OSTATE は特に必要がなければ省略することもできます。

この typedef による型定義は、typedef を無視すると OState という名前の変数の宣言のように見えることに注意してください。そのように見たときの OState という変数の型が、typedef で定義される型名 OState の意味するデータ型となります。たとえば、

```
typedef struct OSTATE OHistory[100];
```

と書くと、OHistory は「100 個の struct OSTATE 型の要素からなる配列型」を意味する型名となります。つまり、この型定義以降で、たとえば

```
OHistory h;
```

のように h を宣言すると、

```
struct OSTATE h[100];
```

と宣言したのと等価になります。

4 構造体を使ったオセロゲームのプログラム

次のプログラム othello08s.c は、第 3 回の演習問題のプログラム othello08.c[9] を構造体を使って書き直したものです。typedef によって、オセロゲームの局面を表す構造体型を OState と型定義して使っています。関数 main の中で OState 型の変数 state を用意し、この変数のアドレスを他の関数へ渡し、現在の局面にアクセスしてもらうことで局面の進行を実現しています。othello08.c では関数 main の中で行っていた手番 (変数 turn) の初期化は、関数 reset (23 行目) に移動しています。また、同じく main で行っていた (石を置いた後の) 手番の交代は関数 inputMove (83 行目) へ移動しました。

othello08s.c

```
 1 #include <stdio.h>
 2
 3 #define BLACK           (1)         /* 黒石を表す定数 */
 4 #define WHITE           (-BLACK)    /* 白石を表す定数 */
 5 #define NONE            (0)         /* 石がないことを表す */
 6 #define OPPONENT(t)     (-(t))      /* t の相手 */
 7 #define BOARD_SIZE      (8)         /* オセロボードの縦横のマス数 */
 8
 9 #define FOR_EACH(x)             for ((x) = 0; (x) < BOARD_SIZE; (x)++)
10 #define FOR_EACH_POS(x, y)      FOR_EACH(x) FOR_EACH(y)
11
12 typedef struct {
13     int board[BOARD_SIZE][BOARD_SIZE];      /* 盤面の状況 */
14     int turn;                               /* 次の手番 */
15 } OState;
16
17 /* 盤面を初期化する */
18 void reset(OState *state)
19 {
20     int x, y;
21
22     /* 先手は黒 */
23     state->turn = BLACK;
24     /* すべてのマスをクリアする */
25     FOR_EACH_POS(x, y)
```

[9] 第 4 回の付録 11 ページ参照

```
26          state->board[x][y] = NONE;
27      /* 中央に4つの石を置く */
28      state->board[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
29      state->board[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
30      state->board[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
31      state->board[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
32  }
33
34  /* 局面を表示する */
35  void showBoard(OState *state)
36  {
37      int x, y;
38
39      /* 列番号を表示する */
40      FOR_EACH(x)
41          printf(" %d", x+1);
42      printf("\n");
43      /* 各行を表示する */
44      FOR_EACH(y) {
45          printf("%d", y+1);
46          FOR_EACH(x) {
47              if (state->board[x][y] == BLACK)
48                  printf("●");
49              else if (state->board[x][y] == WHITE)
50                  printf("○");
51              else
52                  printf("□");
53          }
54          printf("\n");
55      }
56  }
57
58  /* ユーザーが指定した位置に石を置く */
59  void inputMove(OState *state)
60  {
61      int x, y;
62
63      while (1) {
64          /* 列番号を入力する */
65          do {
66              x = 0;
67              printf("列 = ");
68              scanf("%d", &x);
69          } while (x <= 0 || BOARD_SIZE < x);
70          /* 行番号を入力する */
71          do {
72              y = 0;
73              printf("行 = ");
74              scanf("%d", &y);
75          } while (y <= 0 || BOARD_SIZE < y);
76          if (state->board[x-1][y-1] == NONE)
77              break;
78          printf("そこには置けません\n");
79      }
80      /* 石を置く */
81      state->board[x-1][y-1] = state->turn;
82      /* 手番を変える */
83      state->turn = OPPONENT(state->turn);
84  }
85
86  int main()
87  {
```

```
88      OState state;
89
90      reset(&state);
91      while (1) {
92          showBoard(&state);
93          inputMove(&state);
94      }
95      return 0;
96  }
```

5 演習問題

5.1 othello16.c を構造体を使ったプログラムに変更する

まず前回の演習問題で作成したプログラム othello16.c を、othello17.c にコピーしなさい。つぎに、講義で解説したプログラム othello08s.c をまねて、othello17.c を書き換え、構造体を使ったプログラムに変更しなさい。このとき、オセロゲームの局面を表わすデータ型は、othello08s.c と同じく、

```
typedef struct {
    int board[BOARD_SIZE][BOARD_SIZE];      /* 盤面の状況 */
    int turn;                               /* 次の手番 */
} OState;
```

のように OState を型定義して用いなさい。また、付録 (7 – 10ページ) のプログラムの 156 行目で行っているような手番の初期化は、関数 reset の中で行うようにし、 163 行目の手番の交代の処理は、関数 putDisk の中で行うようにしなさい。othello17.c の関数 main の定義は次のようになります。

othello17.c の main

```
int main()
{
    OState state;

    reset(&state);
    while (1) {
        showBoard(&state);
        if (inputMove(&state))
            break;
    }
    return 0;
}
```

完成したら「mprog1 othello17.c」のようにして mprog1 コマンドを実行し、正しくできているかどうかチェックしてください。正しくできている場合は、そのプログラムが提出されたことになります。プログラムは Prog1 というディレクトリに作ることを忘れないようにしましょう。

次回の予定

次回は othello17.c をさらに発展させて、石を挟むことのできないマスには、プレーヤーが石を置けないようにします。

プログラミングおよび実習 I・第 7 回・終り

付録: 前回の演習問題 othello16.c のプログラム例

othello16.c

```
 1 #include <stdio.h>
 2
 3 #define BLACK           (1)         /* 黒石を表す定数 */
 4 #define WHITE           (-BLACK)    /* 白石を表す定数 */
 5 #define NONE            (0)         /* 石がないことを表す */
 6 #define OPPONENT(t)     (-(t))      /* t の相手 */
 7 #define BOARD_SIZE      (8)         /* オセロボードの縦横のマス数 */
 8
 9 #define IS_VALID(x)             (0 <= (x) && (x) < BOARD_SIZE)
10 #define IS_VALID_POS(x, y)      (IS_VALID(x) && IS_VALID(y))
11 #define FOR_EACH(x)             for ((x) = 0; (x) < BOARD_SIZE; (x)++)
12 #define FOR_EACH_POS(x, y)      FOR_EACH(x) FOR_EACH(y)
13
14 /* キーボードからの文字を改行文字まで読み込み、大きさ size の char 型
15    配列 buf へ格納する。 改行文字や size 番目以降の文字は無視される。
16    buf は必ず NUL 文字で終端される。 ただし、size は 1 以上の整数でな
17    ければならない。関数 getLine の戻り値は、buf へ格納した文字(byte)
18    数となる。*/
19 int getLine(char buf[], int size)
20 {
21     int ch;
22     int n = 0;
23
24     /* 文字が読み込めなかったり、改行文字の場合は繰り返しをやめる */
25     while ((ch = getchar()) != EOF && ch != '\n') {
26         /* 配列 buf にまだ余裕があれば、読み込んだ文字を格納する */
27         if (n < size - 1)
28             buf[n++] = ch;
29     }
30     /* buf を NUL 文字で終端する */
31     buf[n] = '\0';
32     return n;
33 }
34
35 /* 盤面を初期化する */
36 void reset(int b[][BOARD_SIZE])
37 {
38     int x, y;
39
40     /* すべてのマスをクリアする */
41     FOR_EACH_POS (x, y)
42         b[x][y] = NONE;
43     /* 中央に4つの石を置く */
44     b[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
45     b[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
46     b[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
47     b[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
48 }
49
50 /* 局面を表示する */
51 void showBoard(int b[][BOARD_SIZE])
52 {
53     int x, y;
54
55     /* 列番号を表示する */
56     FOR_EACH (x)
57         printf(" %c", x + 'a');
```

```
 58     printf("\n");
 59     /* 各行を表示する */
 60     FOR_EACH (y) {
 61         printf("%c", y + '1');
 62         FOR_EACH(x) {
 63             if (b[x][y] == BLACK)
 64                 printf("●");
 65             else if (b[x][y] == WHITE)
 66                 printf("○");
 67             else
 68                 printf("□");
 69         }
 70         printf("\n");
 71     }
 72 }
 73
 74 /* (x, y) に石を置いた時に、(dx, dy) 方向で挟まれる石を反転する */
 75 void checkDir(int b[][BOARD_SIZE], int t, int x, int y, int dx, int dy)
 76 {
 77     int count = 0;      /* 挟まれた石の数 */
 78
 79     /* (dx, dy) 方向に石を調べていく */
 80     while (1) {
 81         /* 1つ進む */
 82         x += dx;
 83         y += dy;
 84         /* 盤の外や空きマスに至ったら終り */
 85         if (!IS_VALID_POS(x, y) || b[x][y] == NONE)
 86             return;
 87         /* 自分の石なら繰り返しを終了 */
 88         if (b[x][y] == t)
 89             break;
 90         count++;
 91     }
 92     /* 1マスずつ戻りながら石を反転させていく */
 93     while (count-- > 0) {
 94         /* 1つ戻る */
 95         x -= dx;
 96         y -= dy;
 97         /* 石を自分のものにする */
 98         b[x][y] = t;
 99     }
100 }
101
102 /* (x, y) の位置に t の石を置く */
103 void putDisk(int b[][BOARD_SIZE], int t, int x, int y)
104 {
105     int dx, dy;
106
107     for (dx = -1; dx <= 1; dx++)
108         for (dy = -1; dy <= 1; dy++)
109             if (dx != 0 || dy != 0)
110                 checkDir(b, t, x, y, dx, dy);
111     b[x][y] = t;
112 }
113
114 /* ユーザーが指定した位置に石を置く */
115 int inputMove(int b[][BOARD_SIZE], int t)
116 {
```

```
117     char buf[100], ch1, ch2, ch3;
118     int n, x, y;
119
120     /* 位置を入力する */
121     while (1) {
122         printf("%s番 = ", (t == BLACK) ? "黒" : "白");
123         /* キーボードからの1行分の入力を buf に格納する */
124         if (getLine(buf, sizeof buf) < 1)
125             continue;
126         /* buf から空白類を無視して、最大3文字を抜き出す */
127         n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
128         /* q の1文字なら 1 を返り値として return する */
129         if (n == 1 && ch1 == 'q')
130             return 1;
131         /* 2文字でないなら再入力を促す */
132         if (n != 2)
133             continue;
134         /* とりあえず c5 のような形式の入力と仮定してみる */
135         x = ch1 - 'a';
136         y = ch2 - '1';
137         if (IS_VALID_POS(x, y) && b[x][y] == NONE)
138             break;
139         /* 逆に 5c のような形式の入力と仮定してみる */
140         x = ch2 - 'a';
141         y = ch1 - '1';
142         if (IS_VALID_POS(x, y) && b[x][y] == NONE)
143             break;
144         printf("そこには置けません\n");
145     }
146     /* 石を置く */
147     putDisk(b, t, x, y);
148     return 0;
149 }
150
151 int main()
152 {
153     /* 盤面の状態 (BLACK/WHITE/NONE) */
154     int board[BOARD_SIZE][BOARD_SIZE];
155     /* つぎの手番 */
156     int turn = BLACK;
157
158     reset(board);
159     while (1) {
160         showBoard(board);
161         if (inputMove(board, turn))
162             break;
163         turn = OPPONENT(turn);
164     }
165
166     /* プログラムを終了する */
167     return 0;
168 }
```