プログラミングおよび実習 I・第 8 回

2003 年 6 月 4 日

1　相手の石を挟めるマスにしか石を置けないようにする

オセロゲームのルールでは、プレーヤーは空きマスならどこにでも石を置いてよいのではなく、相手の石を少なくとも 1 つは挟むような位置に石を置かなければなりません[1]。そこで、これまでのプログラムを改良して、このルールに反する置き方ができないようにしましょう。

前回の演習問題の解答例 (付録 8 – 6 ページ) のプログラム `othello17.c` では、関数 `putDisk` から関数 `checkDir` を呼び出して、挟まれた石をすべて反転させた後、次の 4 行のようにして、ユーザーが指定した位置 (x, y) に石を置くとともに、手番を交代させています。

```
118     /* (x, y) に手番の石を置く */
119     state->board[x][y] = state->turn;
120     /* 手番を変える */
121     state->turn = OPPONENT(state->turn);
```

ルールどおりにするためには、挟まれる石が 1 つもなかった場合に、この 2 つの仕事をしないようにすればよいはずです。

そのためには、まず、関数 `checkDir` の定義を変更して、挟まれた相手の石の数を `int` 型の返り値として呼び出し元へ戻すようにします。関数 `putDisk` の側では、`int` 型の変数 (たとえば) `count` を 0 で初期化しておき、8 方向それぞれに対して `checkDir` を呼び出す際に、その返り値 (その方向で挟まれた相手の石の数) を、この変数 `count` へ足し込んでいくようにします。8 つの方向に対する `checkDir` の呼び出しが終った時には、変数 `count` には反転した相手の石の総数が残るはずですから、この値を調べて 0 より大きい時にだけ、(x, y) に石を置く処理 (119 行目) と、手番を変える処理 (121 行目) を行います。

ただし、挟まれて反転する相手の石がない場合、プログラムとしては、もう一度ユーザーに石を置く位置を入力してもらわないといけません。関数 `putDisk` は `inputMove` から呼び出されますので、指定された位置に石を置くことができたかどうかを `putDisk` の返り値として戻しましょう。たとえば、そこに石を置くことができたのなら、`putDisk` はその時挟まれて反転した相手の石の個数を `int` 型の返り値として返し、石を置くことができなかったのなら 0 を返すことにしておきます。関数 `inputMove` では、関数 `putDisk` を呼び出した際に、その返り値を調べ、0 の時には「そこには置けません」というメッセージを表示して、もう一度ユーザーに入力促すようにします。

付録 (8 – 6 ページ) の `othello17.c` の `inputMove` は

```
125 int inputMove(OState *state)
126 {
127     char buf[100], ch1, ch2, ch3;
128     int n, x, y;
          ⋮
131     while (1) {
132         printf("%s番 = ", (state->turn == BLACK) ? "黒" : "白");
              ⋮
144         /* とりあえず c5 のような形式の入力と仮定してみる */
145         x = ch1 - 'a';
```

[1] それができなければ自分の手番をパスしなければなりません

```
146         y = ch2 - '1';
147         if (IS_VALID_POS(x, y) && state->board[x][y] == NONE)
148             break;
149         /* 逆に 5c のような形式の入力と仮定してみる */
150         x = ch2 - 'a';
151         y = ch1 - '1';
152         if (IS_VALID_POS(x, y) && state->board[x][y] == NONE)
153             break;
154         printf("そこには置けません\n");
155     }
156     /* 石を置く */
157     putDisk(state, x, y);
158     return 0;
159 }
```

のようなプログラムになっていますが、ユーザーが入力した位置 (x, y) がゲーム盤の中であるかどうか (IS_VALID_POS(x, y)) とか、空きマスであるかどうか (state->board[x][y] == NONE) についても関数 putDisk で調べることにして[2]、inputMove の定義を次の例のようにするとすっきりします。

```
125 int inputMove(OState *state)
126 {
127     char buf[100], ch1, ch2, ch3;
128     int n;
          ⋮
131     while (1) {
132         printf("%s番 = ", (state->turn == BLACK) ? "黒" : "白");
              ⋮
144         /* とりあえず c5 のような形式の入力と仮定してみる */
145         if (putDisk(state, ch1 - 'a', ch2 - '1') > 0)
146             break;
147         /* 逆に 5c のような形式の入力と仮定してみる */
148         if (putDisk(state, ch2 - 'a', ch1 - '1') > 0)
149             break;
150         printf("そこには置けません\n");
151     }
152     return 0;
153 }
```

2 C での文字処理 (2)

第 4 回では、C で文字を (文字コードとして) 取り扱う方法について解説しました。この節では、文字コードによる文字の種類の判別や、英文字の大文字小文字の変換を行う標準ライブラリ関数[3]を紹介します。これらの関数を利用するためには、前処理指令 #include を使って ctype.h というヘッダファイルを取り込まなければなりません。プログラムの冒頭に

```
#include <ctype.h>
```

の 1 行を書いておきます。利用できる文字処理の関数には以下のようなものがあります。

---

[2] (x, y) がゲーム盤の外だったり、すでに石のあるマスだったりした場合に、関数 putDisk は 0 を返すようにするわけです

[3] 実際には、関数形式のマクロとして ctype.h の中で定義されている場合がほとんどです

関数名 (返り値と引数)	返り値[4]
`int isalnum(int c)`	c が英 (大小) 文字あるいは数字であるかどうか
`int isalpha(int c)`	c が英 (大小) 文字であるかどうか
`int iscntrl(int c)`	c が制御文字であるかどうか
`int isdigit(int c)`	c が数字 (0 〜 9) であるかどうか
`int isgraph(int c)`	c がスペース以外の図形文字であるかどうか
`int islower(int c)`	c が英小文字であるかどうか
`int isprint(int c)`	c が図形文字 (スペースを含む) であるかどうか
`int ispunct(int c)`	c が英文字でも数字でもない図形文字 (スペースを除く) であるかどうか
`int isspace(int c)`	c がスペース、タブ、改行文字であるかどうか
`int isupper(int c)`	c が英大文字であるかどうか
`int isxdigit(int c)`	c が数字か A 〜 F の範囲の英 (大小) 文字であるかどうか
`int tolower(int c)`	c の小文字 (c が英大文字でなければ c そのもの)
`int toupper(int c)`	c の大文字 (c が英小文字でなければ c そのもの)

3 演習問題

3.1 相手の石を挟める位置にしか自分の石を置けないようにする

まず、前回の演習問題で作成したプログラム `othello17.c` を `othello18.c` にコピーしなさい。つぎに、この `othello18.c` を変更して、下の実行例のように、相手の石を少くとも 1 つは挟める位置にしか石を置くことができないようなプログラムにしなさい。講義で解説したように、関数 `checkDir` や `putDisk`、`inputMove` の定義を変更しましょう。

`othello18.c` の実行例

```
s1609h017% ./othello18
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = c3
そこには置けません
黒番 = c4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
```

[4]関数 `is.....` は、条件が真であれば 0 以外の整数を、偽であれば 0 を返します

```
8□□□□□□□□
白番 = e2
そこには置けません
白番 = e3
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□○□□□
4□□●●○□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = q
s1609h017%
```

完成したら「mprog1 othello18.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。次の演習問題も同様です。

3.2 入力の大文字小文字の違いを無視する

まず、プログラム othello18.c を othello19.c にコピーしなさい。つぎに、この othello19.c を変更して、下の実行例のように、ユーザーが入力した文字列中の大文字小文字の違いを無視するようなプログラムにしなさい。ctype.h の関数 tolower あるいは toupper を利用すると簡単です。

othello19.c の実行例

```
s1609h017% ./othello19
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = c2
そこには置けません
黒番 = C4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = 3e
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□○□□□
4□□●●○□□□
```

```
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = Q
s1609h017%
```

次回の予定

次回は `othello19.c` をさらに発展させて、次のようなことを実現する予定です。

- プレーヤーが手番のパスを選択できるようにします。
- ルール違反のパスができないようにします。

プログラミングおよび実習 I・第 8 回・終り

付録: 前回の演習問題 othello17.c のプログラム例

othello16.c から変更された行には * 印が付してあります。

othello17.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 typedef struct {
* 15     int board[BOARD_SIZE][BOARD_SIZE];  /* 盤面の状況 */
* 16     int turn;                           /* 次の手番 */
* 17 } OState;
  18
  19 /* 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
  20    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。
  21    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
  22    ければならない。関数 getLine の戻り値は、buf へ格納した文字(byte)
  23    数となる。*/
  24 int getLine(char buf[], int size)
  25 {
  26     int ch;
  27     int n = 0;
  28
  29     /* 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 */
  30     while ((ch = getchar()) != EOF && ch != '\n') {
  31         /* 配列 buf にまだ余裕があれば、読み込んだ文字を格納する */
  32         if (n < size - 1)
  33             buf[n++] = ch;
  34     }
  35     /* buf を NUL 文字で終端する */
  36     buf[n] = '\0';
  37     return n;
  38 }
  39
  40 /* 盤面を初期化する */
* 41 void reset(OState *state)
  42 {
  43     int x, y;
  44
  45     /* 先手は黒 */
* 46     state->turn = BLACK;
  47     /* すべてのマスをクリアする */
  48     FOR_EACH_POS (x, y)
* 49         state->board[x][y] = NONE;
  50     /* 中央に4つの石を置く */
* 51     state->board[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
* 52     state->board[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
* 53     state->board[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
* 54     state->board[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
  55 }
```

```
  56
  57 /* 盤面を表示する */
* 58 void showBoard(OState *state)
  59 {
  60     int x, y;
  61
  62     /* 列番号を表示する */
  63     FOR_EACH (x)
  64         printf(" %c", x + 'a');
  65     printf("\n");
  66     /* 各行を表示する */
  67     FOR_EACH (y) {
  68         printf("%c", y + '1');
  69         FOR_EACH(x) {
* 70             if (state->board[x][y] == BLACK)
  71                 printf("●");
* 72             else if (state->board[x][y] == WHITE)
  73                 printf("○");
  74             else
  75                 printf("□");
  76         }
  77         printf("\n");
  78     }
  79 }
  80
  81 /* (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 */
* 82 void checkDir(OState *state, int x, int y, int dx, int dy)
  83 {
  84     int count = 0;      /* 挟まれた石の数 */
  85
  86     /* (dx, dy) 方向に石を調べていく */
  87     while (1) {
  88         /* 1つ進む */
  89         x += dx;
  90         y += dy;
  91         /* 盤の外や空きマスに至ったら終り */
* 92         if (!IS_VALID_POS(x, y) || state->board[x][y] == NONE)
  93             return;
  94         /* 自分の石なら繰り返しを終了 */
* 95         if (state->board[x][y] == state->turn)
  96             break;
  97         count++;
  98     }
  99     /* 1マスずつ戻りながら石を反転させていく */
 100     while (count-- > 0) {
 101         /* 1つ戻る */
 102         x -= dx;
 103         y -= dy;
 104         /* 石を自分のものにする */
*105         state->board[x][y] = state->turn;
 106     }
 107 }
 108
 109 /* (x, y) の位置に手番の石を置く */
*110 void putDisk(OState *state, int x, int y)
 111 {
 112     int dx, dy;
 113
 114     for (dx = -1; dx <= 1; dx++)
 115         for (dy = -1; dy <= 1; dy++)
```

```
 116             if (dx != 0 || dy != 0)
*117                 checkDir(state, x, y, dx, dy);
 118     /* (x, y) に手番の石を置く */
*119     state->board[x][y] = state->turn;
 120     /* 手番を変える */
*121     state->turn = OPPONENT(state->turn);
 122 }
 123
 124 /* ユーザーが指定した位置に石を置く */
*125 int inputMove(OState *state)
 126 {
 127     char buf[100], ch1, ch2, ch3;
 128     int n, x, y;
 129
 130     /* 位置を入力する */
 131     while (1) {
*132         printf("%s番 = ", (state->turn == BLACK) ? "黒" : "白");
 133         /* キーボードからの1行分の入力を buf に格納する */
 134         if (getLine(buf, sizeof buf) < 1)
 135             continue;
 136         /* buf から空白類を無視して、最大3文字を抜き出す */
 137         n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
 138         /* q の1文字なら 1 を返り値として return する */
 139         if (n == 1 && ch1 == 'q')
 140             return 1;
 141         /* 2文字でないなら再入力を促す */
 142         if (n != 2)
 143             continue;
 144         /* とりあえず c5 のような形式の入力と仮定してみる */
 145         x = ch1 - 'a';
 146         y = ch2 - '1';
*147         if (IS_VALID_POS(x, y) && state->board[x][y] == NONE)
 148             break;
 149         /* 逆に 5c のような形式の入力と仮定してみる */
 150         x = ch2 - 'a';
 151         y = ch1 - '1';
*152         if (IS_VALID_POS(x, y) && state->board[x][y] == NONE)
 153             break;
 154         printf("そこには置けません\n");
 155     }
 156     /* 石を置く */
*157     putDisk(state, x, y);
 158     return 0;
 159 }
 160
 161 int main()
 162 {
*163     OState state;
 164
*165     reset(&state);
 166     while (1) {
*167         showBoard(&state);
*168         if (inputMove(&state))
 169             break;
*170     }
 171     return 0;
 172 }
```