プログラミングおよび実習 I・第 10 回

2003 年 6 月 18 日

1　局面の状態の一部として白石と黒石の数を保持する

オセロゲームのプログラムがユーザーに入力を促すときに、次の実行例のように現在の局面での黒石と白石の数を表示することを考えます。

実行例

```
s1609h017% ./othello22
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番（黒2/白2）= c4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番（黒4/白1）= c3
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番（黒3/白3）= c2
```

実行例の続き

```
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□□●○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番（黒5/白2）= b4
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□○○○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番（黒4/白4）= a5
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□●○○●□□□
5●□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番（黒6/白3）= q
s1609h017%
```

最も単純な方法は、次のプログラム例のように、関数 inputMove の中で盤面の黒石と白石をすべて数えあげることで実現することです。

```
    Boolean inputMove(OState *state)
    {
        char buf[100], ch1, ch2, ch3;
*       int n, x, y, blackCount = 0, whiteCount = 0;

        /* 黒石と白石の数を数える */
*       FOR_EACH_POS(x, y) {
*           if (state->board[x][y] == BLACK)
*               blackCount++;
*           else if (state->board[x][y] == WHITE)
*               whiteCount++;
*       }
        /* 位置を入力する */
        while (TRUE) {
*           printf("%s番（黒%d/白%d）= ", (state->turn == BLACK) ? "黒" : "白",
*                   blackCount, whiteCount);
            /* キーボードからの1行分の入力を buf に格納する */
            if (getLine(buf, sizeof buf) < 1)
```

```
            continue;
        ⋮
```

しかし、黒石や白石の個数が必要となる場面は他にもありそうですし[1]、その度に石を数え上げるのは面倒ですから、局面の状態の一部として、常に黒石の数と白石の数を保持するようにしてみましょう。

局面の状態を表現している構造体 (OState 型) に、その新しいメンバとして、黒石の数 blackCount と白石の数 whiteCount の 2 つを加え、

```
typedef struct {
    int board[BOARD_SIZE][BOARD_SIZE];  /* 盤面の状況 */
    int turn;                           /* 次の手番 */
    int blackCount;                     /* 黒石の数 */
    int whiteCount;                     /* 白石の数 */
} OState;
```

のような定義に変更します。ゲーム中の (現在の) 局面は、関数 main の中で宣言 (定義) されている変数 state に保持されていますから、この変数の blackCount と whiteCount の 2 つのメンバの値が、常に (現在の) 黒石の数と白石の数となるように調整します。そのために必要な処理は次の 2 つです。

1. 関数 reset で局面を初期化するときに、blackCount と whiteCount をともに 2 で初期化する。
2. 関数 putDisk が、最後の引数 update を TRUE として呼び出されたときには、反転した石の個数 count を利用して、blackCount や whiteCount の値を更新する。たとえば、黒番で putDisk が呼ばれた時には、blackCount に count (反転して黒石になった石の分) + 1 (今置いた黒石の分) を加え、逆に、whiteCount は count だけ減ずるようにし、白番で呼ばれたのなら、whiteCount に count + 1 を加え、blackCount は count だけ減ずるようにする。

このようにしておけば、関数 inputMove では、入力を促すプロンプトの表示部分を

```
printf("%s番 (黒%d/白%d) = ", (state->turn == BLACK) ? "黒" : "白",
       state->blackCount, state->whiteCount);
```

とするだけで済み、黒石や白石の数を毎回数える必要はなくなります。

2 終局の判定を行う

2 人のプレーヤーがどちらも石を打つことができなくなったときにオセロゲームは終局します。ゲームが終局に至ったら、勝ち負けを表示してプログラムを終了させることにしましょう。勝ち負けの表示は、次のように関数 showResult を定義しておき、この関数を呼び出すことで行うことにします。

```
/* 勝ち負けを表示する */
void showResult(OState *state)
{
    if (state->blackCount > state->whiteCount)
        printf("黒の勝ちです");
    else if (state->whiteCount > state->blackCount)
        printf("白の勝ちです");
    else
        printf("引き分けです");
    printf(" (黒%d/白%d)\n", state->blackCount, state->whiteCount);
}
```

[1] たとえば、ゲームが終って勝ち負けを表示する時にも必要になると思われます

オセロゲームの局面が終局に至るのは次の 2 つの場合が考えられます。

(a) 8×8 のマスがすべて埋め尽された場合。

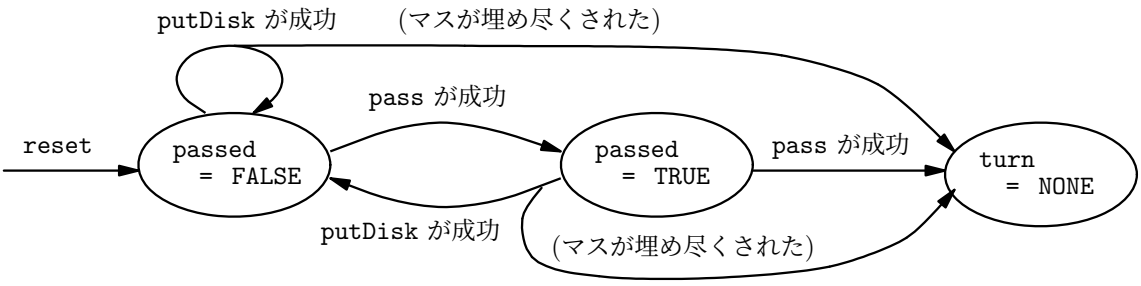
(b) 黒番と白番のプレーヤーが続けてどちらもパスをした (しなければならなくなった) 場合。

ここでは、オセロゲームの局面が終局に至ったことを、局面の状態を表現している構造体 (OState 型) のメンバ turn の値を NONE とすることで表すことにしましょう。

(a) については、関数 putDisk が (最後の引数を TRUE として) 呼び出されて新な石が盤上に置かれた時にチェックすることができるはずです。blackCount と whiteCount の和が BOARD_SIZE の 2 乗に至ったら、すべてのマスが石で埋め尽されたことになります。

(b) については、連続してパスが起きたかどうかを調べなければなりません。そこで、局面を表現している構造体 (OState 型) に新たなメンバ (たとえば passed) を追加して、これを現在の局面がパスの直後の状態であるかどうかを表す**フラグ**として用いることにします。パスが起きた時にフラグを上げ (このメンバを TRUE にし)、関数 putDisk が新な石を置いた時にはフラグを下ろし (FALSE にし) ます。このフラグを上げるのは関数 pass ですが、フラグを上げる前にその時のフラグの状態をチェックし、すでに上がっていれば局面を終局状態 (turn を NONE に変更) します。

以上をまとめると、関数 inputMove から呼び出される putDisk や pass の成功により、ゲームの局面は次の図のように移り変わっていくことになります。



ゲームが終局したことをユーザーに知らせるためには、関数 inputMove の定義を修正します。ユーザーに入力促す前に turn の値をチェックし、turn の値が NONE だったら (終局している状態なら) 関数 showResult を呼び出した後、TRUE を返り値として関数 main に戻るようにすれば十分です[2]。つまり、関数 inputMove の定義には、つぎの * 印を付けた 4 行を追加します。

```
    /* ユーザーが指定した位置に石を置く */
    Boolean inputMove(OState *state)
    {
        char buf[100], ch1, ch2, ch3;
        int n;

        /* 位置を入力する */
        while (TRUE) {
            /* 終局していたら勝ち負けを表示してプログラムを終了する */
*           if (state->turn == NONE) {
*               showResult(state);
*               return TRUE;
*           }
```

[2]同じことは、関数 main の中でも (関数 inputMove を呼び出す前に) できそうですが、プレーヤーやパスをした場合には関数 inputMove から戻る事なく終局に至りますので、inputMove の側で終局のチェックを行う方がよさそうです

```
printf("%s番 (黒%d/白%d) = ", (state->turn == BLACK) ? "黒" : "白",
       state->blackCount, state->whiteCount);
/* キーボードからの1行分の入力を buf に格納する */
if (getLine(buf, sizeof buf) < 1)
    continue;
⋮
```

3 演習問題

3.1 双方の石の個数を表示する

まず、前回の演習問題で作成したプログラム othello21.c を othello22.c にコピーしなさい。つぎに、この othello22.c を変更して、第1節で解説したように、ユーザーに石を置く位置の入力を促す際に、黒石と白石の個数を表示するようにしなさい。石の個数の表示は 10 – 1ページの実行例と同じになるようにしてください。

完成したら「mprog1 othello22.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 終局の処理をする

まず、プログラム othello22.c を othello23.c にコピーしなさい。次に、この othello23.c を変更して、ゲーム盤の大きさが 4×4 となるように変更しなさい。また、さらに othello23.c を変更して、第2節で解説したように、ゲームが終局したら勝ち負けを表示してプログラムが終了するようにしなさい。次の 2 つは othello23.c の実行例です。

実行例 1

```
s1609h017% ./othello23
 a b c d
1□□□□
2□○●□
3□●○□
4□□□□
黒番 (黒2/白2) = a2
 a b c d
1□□□□
2●●●□
3□●○□
4□□□□
白番 (黒4/白1) = a3
 a b c d
1□□□□
2●●●□
3○○○□
4□□□□
黒番 (黒3/白3) = a4
 a b c d
1□□□□
2●●●□
3●●○□
4●□□□
```

続き

```
白番 (黒6/白1) = c1
 a b c d
1□□○□
2●●○□
3●●○□
4●□□□
黒番 (黒5/白3) = d1
 a b c d
1□□○●
2●●●□
3●●○□
4●□□□
白番 (黒7/白2) = a1
 a b c d
1○□○●
2●○●□
3●●○□
4●□□□
黒番 (黒6/白4) = b1
 a b c d
1○●●●
2●●●□
3●●○□
4●□□□
```

続き

```
白番 (黒9/白2) = p
黒番 (黒9/白2) = d4
 a b c d
1○●●●
2●●●□
3●●●□
4●□□●
白番 (黒11/白1) = p
黒番 (黒11/白1) = p
黒の勝ちです (黒11/白1)
s1609h017%
```

実行例 2

```
s1609h017% ./othello23
 a b c d
1□□□□
2□○●□
3□●○□
4□□□□
黒番（黒2/白2）= a2
 a b c d
1□□□□
2●●●□
3□●○□
4□□□□
白番（黒4/白1）= a3
 a b c d
1□□□□
2●●●□
3○○○□
4□□□□
黒番（黒3/白3）= a4
 a b c d
1□□□□
2●●●□
3●●○□
4●□□□
白番（黒6/白1）= c1
 a b c d
1□□○□
2●●○□
3●●○□
4●□□□
```

続き

```
黒番（黒5/白3）= d3
 a b c d
1□□○□
2●●○□
3●●●●
4●□□□
白番（黒7/白2）= c4
 a b c d
1□□○□
2●●○□
3●●○●
4●□○□
黒番（黒6/白4）= d2
 a b c d
1□□○□
2●●●●
3●●○●
4●□○□
白番（黒8/白3）= a1
 a b c d
1○□○□
2●○●●
3●●○●
4●□○□
黒番（黒7/白5）= b1
 a b c d
1○●○□
2●●●●
3●●○●
4●□○□
```

続き

```
白番（黒9/白4）= p
黒番（黒9/白4）= d1
 a b c d
1○●●●
2●●●●
3●●○●
4●□○□
白番（黒11/白3）= p
黒番（黒11/白3）= d4
 a b c d
1○●●●
2●●●●
3●●●●
4●□○●
白番（黒13/白2）= p
黒番（黒13/白2）= b4
 a b c d
1○●●●
2●●●●
3●●●●
4●●●●
黒の勝ちです（黒15/白1）
s1609h017%
```

次回の予定

次回は `othello23.c` をさらに発展させて、`x` の 1 文字を入力すると、プログラムが勝手に石を打ってくれるようにする予定です。

プログラミングおよび実習 I・第 10 回・終り

付録: 前回の演習問題 othello21.c のプログラム例

関数 getLine、reset、showBoard の定義は省略してあります。また、othello19.c から変更された行には * 印が付してあります。

othello21.c

```
   1 #include <stdio.h>
   2 #include <ctype.h>
   3
*  4 typedef int Boolean;
   5
*  6 #define TRUE            (1)
*  7 #define FALSE           (0)
   8
   9 #define BLACK           (1)         /* 黒石を表す定数 */
  10 #define WHITE           (-BLACK)    /* 白石を表す定数 */
  11 #define NONE            (0)         /* 石がないことを表す */
  12 #define OPPONENT(t)     (-(t))      /* t の相手 */
  13 #define BOARD_SIZE      (8)         /* オセロボードの縦横のマス数 */
  14
  15 #define IS_VALID(x)             (0 <= (x) && (x) < BOARD_SIZE)
  16 #define IS_VALID_POS(x, y)      (IS_VALID(x) && IS_VALID(y))
  17 #define FOR_EACH(x)             for ((x) = 0; (x) < BOARD_SIZE; (x)++)
  18 #define FOR_EACH_POS(x, y)      FOR_EACH(x) FOR_EACH(y)
  19
  20 typedef struct {
  21     int board[BOARD_SIZE][BOARD_SIZE];  /* 盤面の状況 */
  22     int turn;                           /* 次の手番 */
  23 } OState;

   ⋮

  87 /* (x, y) に石を置いた時に、(dx, dy) 方向で挟まれる石の数を返す。
  88    update が真なら、挟まれた石を反転する */
* 89 int checkDir(OState *state, int x, int y, int dx, int dy, Boolean update)
  90 {
  91     int count = 0;      /* 挟まれた石の数 */
  92     int i;
  93
  94     /* (dx, dy) 方向に石を調べていく */
* 95     while (TRUE) {
  96         /* 1つ進む */
  97         x += dx;
  98         y += dy;
  99         /* 盤の外や空きマスに至ったら終り */
 100         if (!IS_VALID_POS(x, y) || state->board[x][y] == NONE)
 101             return 0;
 102         /* 自分の石なら繰り返しを終了 */
 103         if (state->board[x][y] == state->turn)
 104             break;
 105         count++;
 106     }
*107     if (update) {
 108         /* 1マスずつ戻りながら石を反転させていく */
 109         for (i = 0; i < count; i++) {
 110             /* 1つ戻る */
 111             x -= dx;
 112             y -= dy;
```

```
 113             /* 石を自分のものにする */
 114             state->board[x][y] = state->turn;
 115         }
*116     }
 117     /* 反転した石の数を返す */
 118     return count;
 119 }
 120
 121 /* (x, y) の位置に手番の石を置いたとして、挟まれる相手の石の数を返り
 122    値として返す。update が真で、かつ挟まれた石があれば実際にそこに石を
 123    置き手番を交代する。 */
*124 int putDisk(OState *state, int x, int y, Boolean update)
 125 {
 126     int count = 0;      /* 挟まれた石の数 */
 127     int dx, dy;
 128
 129     /* 盤の外や、すでに石のあるマスなら石を置かずに 0 を返す */
 130     if (!IS_VALID_POS(x, y) || state->board[x][y] != NONE)
 131         return 0;
 132     for (dx = -1; dx <= 1; dx++)
 133         for (dy = -1; dy <= 1; dy++)
 134             if (dx != 0 || dy != 0)
 135                 count += checkDir(state, x, y, dx, dy, update);
 136     /* update が真で、石が挟まれたのなら石を置き手番を変える */
*137     if (update && count > 0) {
 138         state->board[x][y] = state->turn;
 139         state->turn = OPPONENT(state->turn);
 140     }
 141     /* 反転した石の数を返す */
 142     return count;
 143 }
 144
 145 /* 手番をパスして、返り値として 1 を返す。 相手を挟める手がある場合
 146    にはパスをせずに 0 を返す */
*147 Boolean pass(OState *state)
*148 {
*149     int x, y;
*150
*151     FOR_EACH_POS(x, y) {
*152         if (putDisk(state, x, y, FALSE) > 0)
*153             return FALSE;
*154     }
*155     state->turn = OPPONENT(state->turn);
*156     return TRUE;
*157 }
 158
 159 /* ユーザーが指定した位置に石を置く */
*160 Boolean inputMove(OState *state)
 161 {
 162     char buf[100], ch1, ch2, ch3;
 163     int n;
 164
 165     /* 位置を入力する */
*166     while (TRUE) {
 167         printf("%s番 = ", (state->turn == BLACK) ? "黒" : "白");
 168         /* キーボードからの1行分の入力を buf に格納する */
 169         if (getLine(buf, sizeof buf) < 1)
 170             continue;
 171         /* buf から空白類を無視して、最大3文字を抜き出す */
 172         n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
```

```
 173         ch1 = tolower(ch1);
 174         ch2 = tolower(ch2);
 175         /* q の1文字なら 1 を返り値として return する */
 176         if (n == 1 && ch1 == 'q')
*177             return TRUE;
 178         /* p の1文字なら手番をパスする */
*179         if (n == 1 && ch1 == 'p') {
*180             if (!pass(state))
*181                 printf("パスはできません\n");
*182             continue;
*183         }
 184         /* 2文字でないなら再入力を促す */
 185         if (n != 2)
 186             continue;
 187         /* とりあえず c5 のような形式の入力と仮定してみる */
*188         if (putDisk(state, ch1 - 'a', ch2 - '1', TRUE) > 0)
 189             break;
 190         /* 逆に 5c のような形式の入力と仮定してみる */
*191         if (putDisk(state, ch2 - 'a', ch1 - '1', TRUE) > 0)
 192             break;
 193         printf("そこには置けません\n");
 194     }
*195     return FALSE;
 196 }
 197
 198 int main()
 199 {
 200     OState state;
 201
 202     reset(&state);
*203     do {
*204         showBoard(&state);
*205     } while (!inputMove(&state));
 206     return 0;
 207 }
```