プログラミングおよび実習 I・第 9 回

2003 年 6 月 11 日

1　手番のパスができるようにする

オセロゲームのルールによると、相手の石を挟むことができない場合には自分の手番をパスしなければなりませんし、それ以外の場合にパスをすることはできません。そこで、これまでのプログラムを改良して、プレーヤーがパスを選択できるようにしましょう。ただし、ルールに反するパスはできないようにするのが目標です。

1.1　p の入力でパスが選択できるようにする

まずは、そのパスがルールに従ったものかどうかのチェックはせずに、ユーザーがキーボードから p あるいは P の 1 文字を入力した場合に、手番がパスされるようにしましょう。

パスが選択された場合に行うべき処理は、単に手番を変えることだけですので、関数 inputMove から次のような関数 pass を呼び出すことによって実現できます。

```
void pass(OState *state)
{
    state->turn = OPPONENT(state->turn);
}
```

この時 inputMove の定義には、* を付けた次の 4 行を加えます。

```
int inputMove(OState *state)
{
    ⋮
    while (1) {
        printf("%s番 = ", (state->turn == BLACK) ? "黒" : "白");
        ⋮
        /* q の1文字なら 1 を返り値として return する */
        if (n == 1 && ch1 == 'q')
            return 1;
        /* p の1文字なら手番をパスする */
*       if (n == 1 && ch1 == 'p') {
*           pass(state);
*           continue;
*       }
        /* 2文字でないなら再入力を促す */
        if (n != 2)
            continue;
        ⋮
    }
    return 0;
}
```

1.2　ルール違反のパスができないようにする

ただし、本来のオセロのルールではいつでもパスができるわけではありません。相手を挟めるような手が残っていないかどうかを調べる必要があります。

前回の演習問題の解答例 (付録 9 – 5 ページ) のプログラム othello19.c では、関数 checkDir や関数 putDisk の定義を変更して、(石を置いたときに) 挟まれて反転した相手の石の数を数えられるよう

にしました。これらの関数を、**石を実際に反転することなしに**挟まれる相手の石の数を数えられるようにできれば、pass や inputMove の関数定義を

```
* int pass(OState *state)
  {
*     int x, y;

*     FOR_EACH_POS(x, y) {
*         if (putDisk(state, x, y, 0) > 0)
*             return 0;
*     }
      state->turn = OPPONENT(state->turn);
*     return 1;
  }

  int inputMove(OState *state)
  {
      ⋮
      while (1) {
          ⋮
          /* p の1文字なら手番をパスする */
*         if (n == 1 && ch1 == 'p') {
*             if (!pass(state))
*                 printf("パスはできません\n");
*             continue;
*         }
          /* 2文字でないなら再入力を促す */
          if (n != 2)
              continue;
          /* とりあえず c5 のような形式の入力と仮定してみる */
*         if (putDisk(state, ch1 - 'a', ch2 - '1', 1) > 0)
              break;
          /* 逆に 5c のような形式の入力と仮定してみる */
*         if (putDisk(state, ch2 - 'a', ch1 - '1', 1) > 0)
              break;
          ⋮
      }
      return 0;
  }
```

のように変更することでこれが実現できるはずです[1]。関数 putDisk には int 型の引数が 1 つ追加されており、関数 pass からは、この引数を 0 として、また、inputMove からは 1 として呼び出しています。putDisk の関数定義の詳細は割愛しますが、この引数の値が 0 でないときの動作はこれまでと同じにしておき、 0 のときには、現在の局面は全く変更することなしに、挟まれる石の数だけを数えて返り値として返すようにしておきます[2]。これまでの putDisk は checkDir を 8 つの方向について呼び出すことで実現されていますので、checkDir にも引数を追加して、局面を変更せず (実際に石を反転することなし) に、その方向で挟れる相手の石を数を数えることができるようにしておく必要があります。

関数 pass では、盤上のすべてのマスに関して、関数 putDisk を (最後の引数を 0 として) 呼び出し、その返り値を調べることで、相手の石を挟める手があるかどうかを調べています。どのマスに置いたとしても相手を挟むことができない場合は手番を交替しています (返り値は 1) が、相手を挟むことの

[1]前節のプログラムからの変更部分には * が付けられています

[2]つまり、挟まれた石を反転したり、 (x, y) に手番の石を置いて手番を交替したりせずに、挟まれる石の数を調べます

できるマスを発見した場合は何もせずに呼び出し元へ戻っています (返り値は 0)。関数 inputMove では pass の返り値を調べることにより、パスができたかどうかを判定しています。

2 演習問題

2.1 p でパスができるようにする

まず、前回の演習問題で作成したプログラム othello19.c を othello20.c にコピーしなさい。つぎに、この othello20.c を変更して、1.1節で解説したように、ユーザーが p あるいは P の 1 文字を入力するといつでもパスができるようにしなさい。次は othello20.c の実行例です。相手の石を挟む手が残っている場合でもパスができていることに注意してください。

othello20.c の実行例

```
s1609h017% ./othello20
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = p
白番 = P
黒番 = c4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = c3
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = p
```

実行例の続き

```
白番 = c5
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□○○●□□□
5□□○○○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = c6
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□○○●□□□
5□□○●○□□□
6□□●□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = p
黒番 = b4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□●●●●□□□
5□□○●○□□□
6□□●□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = q
s1609h017%
```

完成したら「mprog1 othello20.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。

2.2 ルール違反のパスができないようにする

まず、プログラム othello20.c を othello21.c にコピーしなさい。つぎに、この othello21.c を変更して、1.2節で解説したように、ユーザーがルール違反のパスができないようにしなさい。次は othello21.c の実行例です。

実行例

```
s1609h017% ./othello21
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = p
パスはできません
黒番 = c4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = P
パスはできません
白番 = C3
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = c2
 a b c d e f g h
```

続き 1

```
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□□●○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = b4
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□○○○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = a5
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□●○○●□□□
5●□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = f4
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□●○○○○□□
5●□□●○□□□
6□□□□□□□□
```

続き 2

```
7□□□□□□□□
8□□□□□□□□
黒番 = g4
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□●●●●●●□
5●□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = c5
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□●●●●●●□
5●□○○○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = d6
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□●□□□□□
4□●●●●●●□
5●□●●●□□□
6□□□●□□□□
7□□□□□□□□
8□□□□□□□□
白番 = p
黒番 = p
白番 = q
s1609h017%
```

次回の予定

次回は othello21.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。

- 局面を表示する際に、それぞれのプレーヤーの石の数を表示するようにします。
- すべてのマスが石で埋め尽くされたら、終局して勝ち負けを表示するようにします。
- 2 人のプレーヤーがどちらもパスをした場合も同様にします。

プログラミングおよび実習 I・第 9 回・終り

付録: 前回の演習問題 othello19.c のプログラム例

関数 getLine、reset、showBoard、main の定義は省略してあります。また、othello17.c から変更された行には * 印が付してあります。

othello19.c

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

   ⋮

  82 /* (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。
  83    関数 checkDir の返り値は、反転した石の数となる。 */
* 84 int checkDir(OState *state, int x, int y, int dx, int dy)
  85 {
  86     int count = 0;      /* 挟まれた石の数 */
* 87     int i;
  88
  89     /* (dx, dy) 方向に石を調べていく */
  90     while (1) {
  91         /* 1つ進む */
  92         x += dx;
  93         y += dy;
  94         /* 盤の外や空きマスに至ったら終り */
  95         if (!IS_VALID_POS(x, y) || state->board[x][y] == NONE)
* 96             return 0;
  97         /* 自分の石なら繰り返しを終了 */
  98         if (state->board[x][y] == state->turn)
  99             break;
 100         count++;
 101     }
 102     /* 1マスずつ戻りながら石を反転させていく */
*103     for (i = 0; i < count; i++) {
 104         /* 1つ戻る */
 105         x -= dx;
 106         y -= dy;
 107         /* 石を自分のものにする */
 108         state->board[x][y] = state->turn;
 109     }
 110     /* 反転した石の数を返す */
*111     return count;
 112 }
 113
```

```
 114 /* (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
 115    値として返す。挟れた石があれば実際にそこに石を置き手番を交代する。 */
*116 int putDisk(OState *state, int x, int y)
 117 {
*118     int count = 0;      /* 挟まれた石の数 */
 119     int dx, dy;
 120
 121     /* 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 */
 122     if (!IS_VALID_POS(x, y) || state->board[x][y] != NONE)
*123         return 0;
 124     for (dx = -1; dx <= 1; dx++)
 125         for (dy = -1; dy <= 1; dy++)
 126             if (dx != 0 || dy != 0)
*127                 count += checkDir(state, x, y, dx, dy);
 128     /* 石が挟まれたのなら石を置き手番を変える */
*129     if (count > 0) {
 130         state->board[x][y] = state->turn;
 131         state->turn = OPPONENT(state->turn);
*132     }
 133     /* 反転した石の数を返す */
*134     return count;
 135 }
 136
 137 /* ユーザーが指定した位置に石を置く */
 138 int inputMove(OState *state)
 139 {
 140     char buf[100], ch1, ch2, ch3;
 141     int n;
 142
 143     /* 位置を入力する */
 144     while (1) {
 145         printf("%s番 = ", (state->turn == BLACK) ? "黒" : "白");
 146         /* キーボードからの1行分の入力を buf に格納する */
 147         if (getLine(buf, sizeof buf) < 1)
 148             continue;
 149         /* buf から空白類を無視して、最大3文字を抜き出す */
 150         n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
*151         ch1 = tolower(ch1);
*152         ch2 = tolower(ch2);
 153         /* q の1文字なら 1 を返り値として return する */
 154         if (n == 1 && ch1 == 'q')
 155             return 1;
 156         /* 2文字でないなら再入力を促す */
 157         if (n != 2)
 158             continue;
 159         /* とりあえず c5 のような形式の入力と仮定してみる */
*160         if (putDisk(state, ch1 - 'a', ch2 - '1') > 0)
*161             break;
 162         /* 逆に 5c のような形式の入力と仮定してみる */
*163         if (putDisk(state, ch2 - 'a', ch1 - '1') > 0)
*164             break;
 165         printf("そこには置けません\n");
 166     }
 167     return 0;
 168 }
```

⋮