プログラミングおよび実習 I・第 6 回

2003 年 5 月 14 日

1　マクロ定義 (2)

前処理指令 `#define` によるマクロ定義は、プログラム中の字句 (の並び) を別の字句 (の並び) に置き換えます[1]。この置き換えのことをマクロ展開と呼びますが、マクロ展開によって得られる字句 (の並び) は、それ自身が完全な式や文の形をしている必要はありません。式や文の一部をマクロ展開によって生成することもできます。たとえば、これまで作成したオセロゲームのプログラムには、

```
for (x = 0; x < BOARD_SIZE; x++)
    ...
```

や

```
for (y = 0; y < BOARD_SIZE; y++)
    ...
```

のような形の `for` 文が何度も出現しますが、

```
#define FOR_EACH(x)       for ((x) = 0; (x) < BOARD_SIZE; (x)++)
```

のように `FOR_EACH` をマクロ定義をしておけば、それぞれ

```
FOR_EACH (x)
    ...
```

や

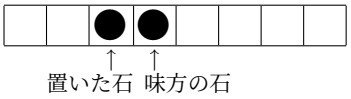
```
FOR_EACH (y)
    ...
```

のように書くことができます。こうするとプログラムが簡潔になるばかりではなく、「`<`」と書くべきところを「`<=`」としてしまうような間違いを減らすことができます。付録 (6 – 7 ページ) のプログラム `othello13.c` の 9 〜 12 行目では、このようなことを目的としたマクロ定義を行っています。

2　挟まれた石を反転するアルゴリズム

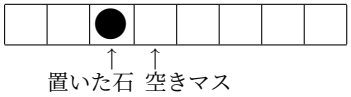
オセロゲームのルールでは、マスに石を置いたとき、その石と味方の石で (隙間なく) 挟まれた相手の石はすべて反転して味方の石に変ります。これをどのような方法で実現すればよいかを考えましょう。挟まれて反転する可能性のある相手の石は、石を置いたマスを起点として、上、右斜め上、右、右斜め下、下、左斜め下、左、左斜め上の 8 つのいずれかの方向に並んでいるはずです。この 8 つの方向それぞれに対して、今置かれた石と味方の石とで隙間なく挟まれている相手の石を反転することができればよいことになります。ここでは、まず、右方向だけに絞って考えてみます。たとえば、黒石があるマスに置かれたとして、そのマスから見て右方向のマスの状況は次の 6 つの場合に分類できるはずです。

(a) 置いた石の右隣が味方の石のとき — 反転する石はない。

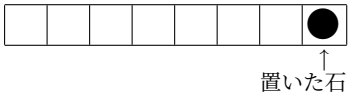


[1]第 1 回配布資料の 11 ページ「2.13 マクロ定義」の節を参照してください

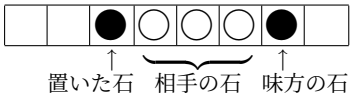
(b) 置いた石の右隣が空きマスのとき — 反転する石はない。



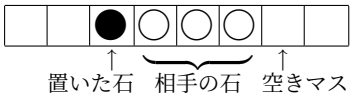
(c) 置いた石の右隣りにマスがないとき — 反転する石はない。



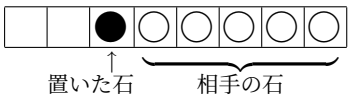
(d) 置いた石の右隣から相手の石が並んでいて、味方の石で終っているとき — 相手の石が反転する。



(e) 置いた石の右隣から相手の石が並んでいて、空きマスで終っているとき — 反転する石はない。



(f) 置いた石の右隣からゲーム盤の端まで相手の石が並んでいるとき — 反転する石はない。



この内、(a)、(b)、(c) については、相手の石の数が 0 個の場合と考えると、それぞれ (d)、(e)、(f) の特別の場合と見なすことができますので、結局、どのような場合であっても、置いた石の右隣から相手の石がいくつか (0 個も可) 並んでおり、その並びが (d) 味方の石か、(e) 空きマスか、(f) ゲーム盤の端で終っていると考えて、(d) の場合だけ相手の石を反転すればよいことになります。そこで、次のような方針で右方向で挟まれる石を反転させることにします。

- 石の置かれたマスの右隣りのマスから、右方向に 1 つマスずつ相手の石の数を数えて行く。
- 空きマスやゲーム盤の外に至ったら何もする必要はない — (e) や (f) の場合に対応。
- 味方の石に至ったら、そこから 1 マスずつ (左方向に) 戻りながら、数えた相手の石の数の分だけ味方の石に変える — (d) の場合に対応。

次は、このような仕事を行う関数 `checkRight` の定義です[2]。関数 `checkRight` は、引数の 2 次元配列 `b` で表現された盤面で、手番 `t` の石が (`x`, `y`) のマスに置かれた時に、そこから右方向で挟まれる相手の石を反転します[3]。関数 `checkRight` は、(`x`, `y`) のマスに `t` の石を置く (`b[x][y]` に `t` を代入する) 仕事自体は行っていません。

関数 checkRight

```
void checkRight(int b[][BOARD_SIZE], int t, int x, int y)
{
    int count = 0;      /* 挟まれた石の数 */

    /* 右方向に石を調べていく */
    while (1) {
```

[2]付録 (6 – 7 ページ) のプログラム 9 〜 10 行目で定義されているマクロ `IS_VALID_POS` が使われています

[3]そのように配列 `b` の内容を書き換えます

```
        /* 1つ進む */
        x++;
        /* 盤の外や空きマスに至ったら終り */
        if (!IS_VALID_POS(x, y) || b[x][y] == NONE)
            return;
        /* 自分の石なら繰り返しを終了 */
        if (b[x][y] == t)
            break;
        count++;
    }
    /* 1マスずつ戻りながら石を反転させていく */
    while (count-- > 0) {
        /* 1つ戻る */
        x--;
        /* 石を自分のものにする */
        b[x][y] = t;
    }
}
```

前回の演習問題で作成したプログラム `othello13.c` に、この関数 `checkRight` の定義を加え、関数 `inputMove` で (ユーザーが指定した位置に) 石を置く際に[4]に「`checkRight(b, t, x, y);`」のようにしてこの関数を呼び出せば、右方向で挟まれる相手の石が反転するようになります。このような変更加えたプログラムを `othello14.c` とすると、その実行例は次のようになります[5]。

othello14 の 実行例

```
s1609h017% ./othello14
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = c4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = c3
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = f5
```

実行例の続き

```
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●●●□□□
5□□□●○●□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = c5
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●●●□□□
5□□○○○●□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = b5
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□○□□□□□
4□□●●●□□□
5□●●●●●□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = q
s1609h017%
```

[4]付録 (6 – 7 ページ) のプログラム 107 行目の `b[x][y] = t;` の部分です

[5]置いた石から右方向で挟まれた相手の石は反転していますが、それ以外の方向で挟まれた石は反転していないことに注意してください

3 演習問題

3.1 例題プログラムを試す

講義で解説した (置いた石から右方向で挟まれる相手の石を反転する) プログラム othello14.c を作成し[6]、コンパイル、実行してみなさい。checkRight の関数定義が新たに加わります。inputMove でユーザーが指定した位置に石を置く際に、checkRight を呼び出して、そこから右方向で挟まれる相手の石を反転するのを忘れないようにしましょう。完成したら、「mprog1 othello14.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。以下の 2 つの演習問題も同様です。

3.2 手番を表示する

まず、othello14.c を othello15.c にコピーしなさい。その後 othello15.c を変更して、次の実行例のように、位置の入力を促すプロンプトを、そのときの手番に応じて「黒番 = 」や「白番 = 」となるようにしなさい。

othello15 の 実行例

```
s1609h017% ./othello15
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = c4
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□●●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番 = c5
```

実行例の続き

```
 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 = q
s1609h017%
```

3.3 挟まれた石をすべて反転する

まず、othello15.c を othello16.c にコピーしなさい。その後 othello16.c を変更して、挟まれた石が (どの方向であっても) すべて反転するようにしなさい。ただし、次のような方針にしたがってこれを実現しなさい。

1. 関数 checkRight の名前を checkDir に変更するとともに、int 型の 2 つの仮引数 dx と dy を追加する。また、この関数の定義を書き換えて、(x, y) のマスに手番 t の石を置いた時に、そこから (dx, dy) 方向で挟まれる石を反転するようにする。ただし、(dx, dy) に渡される実引数は

[6] 前回作成した othello13.c をコピーして変更すると簡単かも知れません

$(1, -1)$、$(1, 0)$、$(1, 1)$、$(0, -1)$、$(0, 1)$、$(-1, -1)$、$(-1, 0)$、$(-1, 1)$ のいずれかであるとし、たとえば、dx $= 1$、dy $= -1$ の場合は、(x, y) から右斜め上方向で挟まれた石を反転する。

2. 次のような関数 putDisk を定義して、関数 inputMove では、2 次元配列 b の内容を直接書き換える代りに、関数 putDisk を呼び出すようにする。

```
void putDisk(int b[][BOARD_SIZE], int t, int x, int y)
{
    checkDir(b, t, x, y, -1, -1);
    checkDir(b, t, x, y, -1, 0);
    checkDir(b, t, x, y, -1, 1);
    checkDir(b, t, x, y, 0, -1);
    checkDir(b, t, x, y, 0, 1);
    checkDir(b, t, x, y, 1, -1);
    checkDir(b, t, x, y, 1, 0);
    checkDir(b, t, x, y, 1, 1);
    b[x][y] = t;
}
```

完成したプログラム othello16.c の実行例は次のようになります。

othello16 の 実行例

```
s1609h017% ./othello16
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番 = d3
 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 = c4
```

実行例の続き

```
 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 = q
s1609h017%
```

関数 putDisk は次のように定義することもできます。

```
void putDisk(int b[][BOARD_SIZE], int t, int x, int y)
{
```

```
    int dx, dy;

    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;
}
```

次回の予定

次回は `othello16.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。

- **構造体**と呼ばれるデータ型を使ってオセロゲームの状態を表現することを考えます。
- 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 6 回・終り

付録: 前回の演習問題 othello13.c のプログラム例

othello13.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 /* 盤面を表示する */
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 /* ユーザーが指定した位置に石を置く */
 75 int inputMove(int b[][BOARD_SIZE], int t)
 76 {
 77     char buf[100], ch1, ch2, ch3;
 78     int n, x, y;
 79
 80     /* 位置を入力する */
 81     while (1) {
 82         printf("位置 = ");
 83         /* キーボードからの1行分の入力を buf に格納する */
 84         if (getLine(buf, sizeof buf) < 1)
 85             continue;
 86         /* buf から空白類を無視して、最大3文字を抜き出す */
 87         n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
 88         /* q の1文字なら 1 を返り値として return する */
 89         if (n == 1 && ch1 == 'q')
 90             return 1;
 91         /* 2文字でないなら再入力を促す */
 92         if (n != 2)
 93             continue;
 94         /* とりあえず c5 のような形式の入力と仮定してみる */
 95         x = ch1 - 'a';
 96         y = ch2 - '1';
 97         if (IS_VALID_POS(x, y) && b[x][y] == NONE)
 98             break;
 99         /* 逆に 5c のような形式の入力と仮定してみる */
100         x = ch2 - 'a';
101         y = ch1 - '1';
102         if (IS_VALID_POS(x, y) && b[x][y] == NONE)
103             break;
104         printf("そこには置けません\n");
105     }
106     /* 石を置く */
107     b[x][y] = t;
108     return 0;
109 }
110
111 int main()
112 {
113     /* 盤面の状態 (BLACK/WHITE/NONE) */
114     int board[BOARD_SIZE][BOARD_SIZE];
115     /* つぎの手番 */
116     int turn = BLACK;
117
118     reset(board);
119     while (1) {
120         showBoard(board);
121         if (inputMove(board, turn))
122             break;
123         turn = OPPONENT(turn);
124     }
125
126     /* プログラムを終了する */
127     return 0;
128 }
```