1. Intro. I'm (hastily) writing this program in order to get some basic experience with a generalization of the game of Reversi (which is popularly called "Othello" in its principal variant).

I consider an $m \times n$ board, whose rows are numbered `1`, `2`, `3`, ..., and whose columns are named `a`, `b`, `c`, ... A cell is identified by letter and digit. (I don't forbid $m > 9$ or $n > 26$; but bugs will show up if those parameters get too big.)

When prompted, the user should provide (on *stdin*) the name of the cell where the next move is to be made, followed by a newline (aka 'return'). The newline can optionally be preceded by '`!`', in which case the contents of the current board will be printed (on *stdout*).

The first four moves simply place pieces on the board; they are made without captures. (They are considered to be moves -3, -2, -1, and 0, so that move 1 will be the normal first move.) In standard Othello we have $m = n = 9$, and the preliminary moves are `d5`, `e5`, `e4`, `d4`; the first move might then be `d3`, in which case the color of `d4` will be reversed.

This program actually allows more than two players. With p players, we start with move $1 - p^2$, and use the first p^2 moves to set up the initial position.

I don't attempt to do anything tricky. Cells of the board contain -1 if they're unoccupied, c if they are occupied and the top color is c. The colors are 0, 1, ..., $p - 1$. The value of $board[i][j]$ is maintained for $0 \le i \le m + 1$ and $0 \le j \le n + 1$, but the boundary cells remain unoccupied.

```
#define m 8   /* this many rows */
#define n 8   /* this many columns */
#define p 2   /* this many players */
#include <stdio.h>
#include <stdlib.h>
  char board[m + 2][n + 2];   /* cells of the current board position */
  int move;   /* the current move number */
  char deli[8] = {−1, −1, −1, 0, 0, 1, 1, 1}, delj[8] = {−1, 0, 1, −1, 1, −1, 0, 1};   /* the eight directions */
  char buffer[8];   /* used for input */
  int total[p];   /* how many pieces show this color? */
  ⟨ Subroutines 3 ⟩;
  void main(void)
  {
    register int i, j, k, l, pass, player;
    for (i = 0; i ≤ m + 1; i++)
      for (j = 0; j ≤ n + 1; j++) board[i][j] = −1;
    for (move = 1 − p ∗ p, player = 0; ; move++, player = (player + 1) % p) {
      for (pass = 0; pass < p; pass++) {
        for (i = 1; i ≤ m; i++)
          for (j = 1; j ≤ n; j++)
            if (islegal(i, j, player)) goto nextmove;
        printf("(player␣%c␣cannot␣move)\n", '0' + player);
        player = (player + 1) % p;
      }
      break;   /* the game is over: p passes in a row */
    nextmove: printf("Move␣%d,␣player␣%c:␣", move, '0' + player);
      fflush(stdout);   /* make sure the user sees the prompt */
      ⟨ Set i and j to the coordinates of the next move 2 ⟩;
      makemove(i, j, player);
      if (buffer[2] ≡ '!') print_board();
    }
    print_board();
  }
```

2. ⟨ Set i and j to the coordinates of the next move 2 ⟩ ≡

```
if (¬fgets(buffer, 8, stdin)) {
  fprintf(stderr, "Unexpected␣end␣of␣input!\n");
  exit(−1);
}
j = buffer[0] − 'a' + 1, i = buffer[1] − '0';
if (i < 1 ∨ i > m ∨ j < 1 ∨ j > n) {
  fprintf(stderr, "Cell␣`%c%c'␣doesn't␣exist!\n", buffer[0], buffer[1]);
  print_board();
  goto nextmove;
}
if (¬islegal(i, j, player)) {
  fprintf(stderr, "No!␣`%c%c'␣isn't␣a␣legal␣move␣for␣%c.\n", buffer[0], buffer[1], '0' + player);
  print_board();
  goto nextmove;
}
```

This code is used in section 1.

3. ⟨ Subroutines 3 ⟩ ≡

```
void print_board(void)
{
  register i, j, k;
  for (k = 0; k < p; k++) total[k] = 0;
  for (i = 1; i ≤ m; i++) {
    for (j = 1; j ≤ n; j++) {
      k = board[i][j];
      if (k ≥ 0) total[k]++;
      printf("%c", k < 0 ? '.' : '0' + k);
    }
    if (i ≡ m)
      for (k = 0; k < p; k++) printf("␣%d", total[k]);
    printf("\n");
  }
}
```

See also sections 4 and 6.

This code is used in section 1.

4. This subroutine decides whether or not it's OK to place a piece of color c in cell (i, j) of the board. We assume (without checking) that $1 \le i \le m$, $1 \le j \le n$, and $0 \le c < p$.

⟨ Subroutines 3 ⟩ +≡

```
int islegal(int i, int j, int c)
{
  register int ii, jj, k, l;
  if (board[i][j] ≥ 0) return 0;      /* already occupied */
  if (move ≤ 0) return 1;      /* we're just gettin' started */
  for (k = 0; k < 8; k++) ⟨ If direction k allows a move, return 1; otherwise continue 5 ⟩;
  return 0;
}
```

5. ⟨If direction k allows a move, **return** 1; otherwise **continue** 5⟩ ≡

```
{
  for (ii = i + deli[k], jj = j + delj[k], l = 0; board[ii][jj] ≥ 0; ii += deli[k], jj += delj[k], l++)
    if (board[ii][jj] ≡ c) goto maybe;
  continue;    /* no occurrences of c in direction k */
maybe: if (l) return 1;    /* yes, that move reverses at least l cells */
  continue;    /* two adjacent c's */
}
```

This code is used in section 4.

6. ⟨Subroutines 3⟩ +≡

```
void makemove(int i, int j, int c)
{
  register int ii, jj, k;

  board[i][j] = c;
  if (move ≤ 0) return;    /* just gettin' started */
  for (k = 0; k < 8; k++) ⟨Do all reversals in direction k 7⟩;
}
```

7. ⟨Do all reversals in direction k 7⟩ ≡

```
{
  for (ii = i + deli[k], jj = j + delj[k]; board[ii][jj] ≥ 0; ii += deli[k], jj += delj[k])
    if (board[ii][jj] ≡ c) goto reverse;
  continue;    /* no occurrences of c in direction k */
reverse: for (ii −= deli[k], jj −= delj[k]; ii ≠ i ∨ jj ≠ j; ii −= deli[k], jj −= delj[k]) board[ii][jj] = c;
}
```

This code is used in section 6.

8. Index.

OTHELLO