November 24, 2020 at 13:24

**1. Data for dancing.** This program creates data suitable for the DANCE routine, given the description of a board to be covered and a set of polyiamond shapes.

The first line of input names all the board positions, in any order. Each position is a two-digit number representing $x$ and $y$ coordinates, or a two-digit number followed by an asterisk; each "digit" is a single character, `0`–`9` or `a`–`z` representing the numbers 0–35. The asterisk denotes a triangle with point down. For example,

```
00 00* 01 10
```

is one way to describe a triangular board, two units on a side.

The second line of input names all the pieces. Each piece name consists of at most three characters; the name should also be distinguishable from a board position. (The program does not check this.)

The remaining lines of input describe the polyiamonds. First comes the name, followed by two integers $s$ and $t$, meaning that the shape should appear in $s$ rotations and $t$ transpositions. Then come two-digit coordinates for each cell of the shape. For example, the line

```
G 6 2 00* 01 01* 10 10* 20
```

describes a hexiamond that can appear in 12 orientations. (See the analogous program for polyominoes.)

```
#define max_pieces 100   /* at most this many shapes */
#define buf_size 36 * 36 * 3 + 8   /* upper bound on line length */
#include <stdio.h>
#include <ctype.h>
  ⟨Global variables 4⟩
  ⟨Subroutines 3⟩;
  main()
  {
    register char *p, *q;
    register int j, k, n, x, y, z;
    ⟨Read and output the board 2⟩;
    ⟨Read and output the piece names 5⟩;
    ⟨Read and output the pieces 6⟩;
  }
```

**2.** **#define** *panic*(*m*)

```
        { fprintf(stderr, "%s!\n%s", m, buf); exit(−1); }
```

⟨ Read and output the board 2 ⟩ ≡

```
  fgets(buf, buf_size, stdin);
  if (buf[strlen(buf) − 1] ≠ '\n') panic("Input␣line␣too␣long");
  bxmin = bymin = 35; bxmax = bymax = 0;
  for (p = buf; *p; p += 3) {
    while (isspace(*p)) p++;
    if (¬*p) break;
    x = decode(*p);
    if (x < 0) panic("Bad␣x␣coordinate");
    y = decode(*(p + 1));
    if (y < 0) panic("Bad␣y␣coordinate");
    if (*(p + 2) ≡ '*') p++, z = 1; else z = 0;
    if (¬isspace(*(p + 2))) panic("Bad␣board␣position");
    if (board[x][y][z]) panic("Duplicate␣board␣position");
    if (x < bxmin) bxmin = x;
    if (x > bxmax) bxmax = x;
    if (y < bymin) bymin = y;
    if (y > bymax) bymax = y;
    board[x][y][z] = 1;
  }
  if (bxmin > bxmax) panic("Empty␣board");
  fwrite(buf, 1, strlen(buf) − 1, stdout);      /* output all but the newline */
```

This code is used in section 1.

**3.** ⟨ Subroutines 3 ⟩ ≡

```
  int decode(c)
      char c;
  {
    if (c ≤ '9') {
      if (c ≥ '0') return c − '0';
    } else if (c ≥ 'a') {
      if (c ≤ 'z') return c + 10 − 'a';
    }
    return −1;
  }
```

See also section 12.

This code is used in section 1.

**4.** ⟨ Global variables 4 ⟩ ≡

```
  char buf[buf_size];
  int board[36][36][2];      /* cells present */
  int bxmin, bxmax, bymin, bymax;      /* used portion of the board */
```

See also section 7.

This code is used in section 1.

**5.** ⟨ Read and output the piece names 5 ⟩ ≡

```
  if (¬fgets(buf, buf_size, stdin)) panic("No␣piece␣names");
  printf("␣%s", buf);      /* just pass the piece names through */
```

This code is used in section 1.

**6.** ⟨ Read and output the pieces 6 ⟩ ≡

```
while (fgets(buf, buf_size, stdin)) {
  if (buf[strlen(buf) − 1] ≠ '\n') panic("Input␣line␣too␣long");
  for (p = buf; isspace(*p); p++) ;
  if (¬*p) panic("Empty␣line");
  for (q = p + 1; ¬isspace(*q); q++) ;
  if (q > p + 3) panic("Piece␣name␣too␣long");
  for (q = name; ¬isspace(*p); p++, q++) *q = *p;
  *q = '\0';
  for (p++; isspace(*p); p++) ;
  s = *p − '0';
  if ((s ≠ 1 ∧ s ≠ 2 ∧ s ≠ 3 ∧ s ≠ 6) ∨ ¬isspace(*(p + 1))) panic("Bad␣s␣value");
  for (p += 2; isspace(*p); p++) ;
  t = *p − '0';
  if ((t ≠ 1 ∧ t ≠ 2) ∨ ¬isspace(*(p + 1))) panic("Bad␣t␣value");
  n = 0;
  xmin = ymin = 35; xmax = ymax = 0;
  for (p += 2; *p; p += 3, n++) {
    while (isspace(*p)) p++;
    if (¬*p) break;
    x = decode(*p);
    if (x < 0) panic("Bad␣x␣coordinate");
    y = decode(*(p + 1));
    if (y < 0) panic("Bad␣y␣coordinate");
    if (*(p + 2) ≡ '*') p++, z = 1; else z = 0;
    if (¬isspace(*(p + 2))) panic("Bad␣board␣position");
    if (n ≡ 36 * 36 * 2) panic("Pigeonhole␣principle␣says␣you␣repeated␣a␣position");
    xx[n] = x, yy[n] = y, zz[n] = z;
    if (x < xmin) xmin = x;
    if (x > xmax) xmax = x;
    if (y < ymin) ymin = y;
    if (y > ymax) ymax = y;
  }
  if (n ≡ 0) panic("Empty␣piece");
  ⟨ Generate the possible piece placements 8 ⟩;
}
```

This code is used in section 1.

**7.** ⟨ Global variables 4 ⟩ +≡

```
char name[4];     /* name of current piece */
int s, t;     /* symmetry type of current piece */
int xx[36 * 36 * 2], yy[36 * 36 * 2], zz[36 * 36 * 2];     /* coordinates of current piece */
int xmin, xmax, ymin, ymax;     /* range of coordinates */
```

**8.** ⟨ Generate the possible piece placements 8 ⟩ ≡

```
while (t) {
  for (k = 1; k ≤ 6; k++) {
    if (k ≤ s) ⟨ Output translates of the current piece 11 ⟩;
    ⟨ Rotate the current piece 10 ⟩;
  }
  ⟨ Transpose the current piece 9 ⟩;
  t−−;
}
```

This code is used in section 6.

**9.** ⟨ Transpose the current piece 9 ⟩ ≡

```
for (j = 0; j < n; j++) {
  z = xx[j];
  xx[j] = yy[j];
  yy[j] = z;
}
z = xmin; xmin = ymin; ymin = z;
z = xmax; xmax = ymax; ymax = z;
```

This code is used in section 8.

**10.** ⟨ Rotate the current piece 10 ⟩ ≡

```
xmin = ymin = 1000; xmax = ymax = −1000;
for (j = 0; j < n; j++) {
  z = xx[j];
  xx[j] = z + yy[j] + zz[j];
  yy[j] = −z;
  zz[j] = 1 − zz[j];
  if (xx[j] < xmin) xmin = xx[j];
  if (xx[j] > xmax) xmax = xx[j];
  if (yy[j] < ymin) ymin = yy[j];
  if (yy[j] > ymax) ymax = yy[j];
}
```

This code is used in section 8.

**11.** ⟨ Output translates of the current piece 11 ⟩ ≡

```
for (x = bxmin − xmin; x ≤ bxmax − xmax; x++)
  for (y = bymin − ymin; y ≤ bymax − ymax; y++) {
    for (j = 0; j < n; j++)
      if (¬board[x + xx[j]][y + yy[j]][zz[j]]) goto nope;
    printf(name);
    for (j = 0; j < n; j++) {
      printf("␣%c%c", encode(x + xx[j]), encode(y + yy[j]));
      if (zz[j]) printf("*");
    }
    printf("\n");
  nope: ;
  }
```

This code is used in section 8.

**12.** ⟨ Subroutines 3 ⟩ +≡

```
char encode(x)
      int x;
{
   if (x < 10) return '0' + x;
   return 'a' - 10 + x;
}
```

## 13. Index.

# POLYIAMONDS