(See https://cs.stanford.edu/~knuth/programs.html for date.)

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

The first 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 second line of input names all the board positions, in any order except that interior junction points must follow a '`|`'. Each position is of the form `Hxy` or `Vxy` or `Ixy`, where `x` and `y` are digits that represent coordinates; each "digit" is a single character, `0`–`9` or `a`–`z` representing the numbers 0–35. Position `Hxy` is the edge from (x,y) to $(x+1,y)$; position `Vxy` is the edge from (x,y) to $(x,y+1)$; position `Ixy` is the interior point (x,y). For example,

```
H01 H11 V10 V11 | I11
```

is one way to describe a board that makes a small cross shape.

The remaining lines of input describe the polysticks. 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 board positions for each cell of the shape. For example, the line

```
C 4 1 H00 V00 I01 V01 H02
```

describes a hexiamond that can appear in 4 orientations. (See the analogous program for polyominoes.)

```
#define max_pieces 100   /* at most this many shapes */
#define buf_size 3 * 36 * 36 * 4 + 10   /* upper bound on line length */
#include <stdio.h>
#include <ctype.h>
  ⟨ Global variables 5 ⟩
  ⟨ Subroutines 4 ⟩;
  main( )
  {
    register char *p, *q;
    register int j, k, n, x, y, z, bar;
    ⟨ Read and output the piece names 2 ⟩;
    ⟨ Read and output the board 3 ⟩;
    ⟨ Read and output the pieces 6 ⟩;
  }
```

2.

```
#define panic(m)
        { fprintf(stderr, "%s!\n%s", m, buf); exit(−1); }
⟨ Read and output the piece names 2 ⟩ ≡
  if (¬fgets(buf, buf_size, stdin)) panic("No␣piece␣names");
  if (buf[strlen(buf) − 1] ≠ '\n') panic("Input␣line␣too␣long");
  fwrite(buf, 1, strlen(buf) − 1, stdout);   /* output all but the newline */
```

This code is used in section 1.

3. ⟨Read and output the board 3⟩ ≡

```
if (¬fgets(buf, buf_size, stdin)) panic("No␣board");
if (buf[strlen(buf) − 1] ≠ '\n') panic("Input␣line␣too␣long");
bxmin = bymin = 35; bxmax = bymax = 0;
for (p = buf, bar = 0; ∗p; p += 4) {
  while (isspace(∗p)) p++;
  if (¬∗p) break;
  if (∗p ≡ '|' ∧ isspace(∗(p + 1))) {
    bar = 1;
    p −= 2;
    continue;
  }
  x = decode(∗(p + 1));
  if (x < 0) panic("Bad␣x␣coordinate");
  y = decode(∗(p + 2));
  if (y < 0) panic("Bad␣y␣coordinate");
  if (¬isspace(∗(p + 3))) panic("Bad␣board␣position");
  if (∗p ≡ 'H' ∧ ¬bar) z = 0;
  else if (∗p ≡ 'V' ∧ ¬bar) z = 2;
  else if (∗p ≡ 'I' ∧ bar) z = 1;
  else panic("Illegal␣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");
printf("␣%s", buf);     /∗ just pass the board names through ∗/
```

This code is used in section 1.

4. ⟨Subroutines 4⟩ ≡

```
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.

5. ⟨Global variables 5⟩ ≡

```
char buf[buf_size];
int board[36][36][3];     /∗ positions present ∗/
int bxmin, bxmax, bymin, bymax;     /∗ used portion of the board ∗/
```

See also section 7.

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 ≠ 4) ∨ ¬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 += 4, n++) {
    while (isspace(∗p)) p++;
    if (¬∗p) break;
    x = decode(∗(p + 1));
    if (x < 0) panic("Bad␣x␣coordinate");
    y = decode(∗(p + 2));
    if (y < 0) panic("Bad␣y␣coordinate");
    if (¬isspace(∗(p + 3))) panic("Bad␣board␣position");
    if (∗p ≡ 'H') z = 0;
    else if (∗p ≡ 'V') z = 2;
    else if (∗p ≡ 'I') z = 1;
    else panic("Illegal␣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 5 ⟩ +≡

```
char name[4];    /∗ name of current piece ∗/
int s, t;    /∗ symmetry type of current piece ∗/
int xx[36 ∗ 36 ∗ 3], yy[36 ∗ 36 ∗ 3], zz[36 ∗ 36 ∗ 3];    /∗ 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 ≤ 4; 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;
  zz[j] = 2 − zz[j];
}
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] = −yy[j];
  if (zz[j] ≡ 2) xx[j]−−;
  yy[j] = z;
  zz[j] = 2 − 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. Interior points don't have to be on the board; they might, for example, lie on the boundary after translation.

```
⟨ 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 (zz[j] ≠ 1 ∧ ¬board[x + xx[j]][y + yy[j]][zz[j]]) goto nope;
      printf(name);
      for (j = 0; j < n; j++)
        if (board[x + xx[j]][y + yy[j]][zz[j]]) {
          printf("␣%c%c%c", codeletter[zz[j]], encode(x + xx[j]), encode(y + yy[j]));
        }
      printf("\n");
    nope: ;
    }
```

This code is used in section 8.

12.

```
⟨ Subroutines 4 ⟩ +≡
  char codeletter[3] = {'H', 'I', 'V'};

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

13. Index.

POLYSTICKS