

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

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. \langle Read and output the board 3 $\rangle \equiv$

```

if ( $\neg$ fgets(buf, buf_size, stdin)) panic("No_board");
if (buf[strlen(buf) - 1]  $\neq$  '\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 ( $\neg$ *p) break;
    if (*p  $\equiv$  '|'  $\wedge$  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 ( $\neg$ isspace(*(p + 3))) panic("Bad_board_position");
    if (*p  $\equiv$  'H'  $\wedge$   $\neg$ bar) z = 0;
    else if (*p  $\equiv$  'V'  $\wedge$   $\neg$ bar) z = 2;
    else if (*p  $\equiv$  'I'  $\wedge$  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. \langle Subroutines 4 $\rangle \equiv$

```

int decode(c)
    char c;
{
    if (c  $\leq$  '9') {
        if (c  $\geq$  '0') return c - '0';
    } else if (c  $\geq$  'a') {
        if (c  $\leq$  'z') return c + 10 - 'a';
    }
    return -1;
}

```

See also section 12.

This code is used in section 1.

5. \langle Global variables 5 $\rangle \equiv$

```

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. \langle Read and output the pieces 6 $\rangle \equiv$

```

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");
   $\langle$  Generate the possible piece placements 8  $\rangle$ ;
}

```

This code is used in section 1.

7. \langle Global variables 5 $\rangle + \equiv$

```

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. \langle Generate the possible piece placements 8 $\rangle \equiv$
while (t) {
 for ($k = 1$; $k \leq 4$; $k++$) {
 if ($k \leq s$) \langle Output translates of the current piece 11 \rangle ;
 \langle Rotate the current piece 10 \rangle ;
 }
 \langle Transpose the current piece 9 \rangle ;
 $t--$;
 }

This code is used in section 6.

9. \langle Transpose the current piece 9 $\rangle \equiv$
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. \langle Rotate the current piece 10 $\rangle \equiv$
 $xmin = ymin = 1000$; $xmax = ymax = -1000$;
for ($j = 0$; $j < n$; $j++$) {
 $z = xx[j]$;
 $xx[j] = -yy[j]$;
 if ($zz[j] \equiv 2$) $xx[j]--$;
 $yy[j] = z$;
 $zz[j] = 2 - zz[j]$;
 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 ( $x = bxmin - xmin$ ;  $x \leq bxmax - xmax$ ;  $x++$ )
  for ( $y = bymin - ymin$ ;  $y \leq bymax - ymax$ ;  $y++$ ) {
    for ( $j = 0$ ;  $j < n$ ;  $j++$ )
      if ( $zz[j] \neq 1 \wedge \neg board[x + xx[j]][y + yy[j]][zz[j]]$ ) 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.

bar: [1](#), [3](#).
board: [3](#), [5](#), [11](#).
buf: [2](#), [3](#), [5](#), [6](#).
buf_size: [1](#), [2](#), [3](#), [5](#), [6](#).
bxmax: [3](#), [5](#), [11](#).
bxmin: [3](#), [5](#), [11](#).
bymax: [3](#), [5](#), [11](#).
bymin: [3](#), [5](#), [11](#).
c: [4](#).
codeletter: [11](#), [12](#).
decode: [3](#), [4](#), [6](#).
encode: [11](#), [12](#).
exit: [2](#).
fgets: [2](#), [3](#), [6](#).
fprintf: [2](#).
fwrite: [2](#).
isspace: [3](#), [6](#).
j: [1](#).
k: [1](#).
main: [1](#).
max_pieces: [1](#).
n: [1](#).
name: [6](#), [7](#), [11](#).
nope: [11](#).
p: [1](#).
panic: [2](#), [3](#), [6](#).
printf: [3](#), [11](#).
q: [1](#).
s: [7](#).
stderr: [2](#).
stdin: [2](#), [3](#), [6](#).
stdout: [2](#).
strlen: [2](#), [3](#), [6](#).
t: [7](#).
x: [1](#), [12](#).
xmax: [6](#), [7](#), [9](#), [10](#), [11](#).
xmin: [6](#), [7](#), [9](#), [10](#), [11](#).
xx: [6](#), [7](#), [9](#), [10](#), [11](#).
y: [1](#).
ymax: [6](#), [7](#), [9](#), [10](#), [11](#).
ymin: [6](#), [7](#), [9](#), [10](#), [11](#).
yy: [6](#), [7](#), [9](#), [10](#), [11](#).
z: [1](#).
zz: [6](#), [7](#), [9](#), [10](#), [11](#).

- ⟨ Generate the possible piece placements 8 ⟩ Used in section 6.
- ⟨ Global variables 5, 7 ⟩ Used in section 1.
- ⟨ Output translates of the current piece 11 ⟩ Used in section 8.
- ⟨ Read and output the board 3 ⟩ Used in section 1.
- ⟨ Read and output the piece names 2 ⟩ Used in section 1.
- ⟨ Read and output the pieces 6 ⟩ Used in section 1.
- ⟨ Rotate the current piece 10 ⟩ Used in section 8.
- ⟨ Subroutines 4, 12 ⟩ Used in section 1.
- ⟨ Transpose the current piece 9 ⟩ Used in section 8.

POLYSTICKS

	Section	Page
Data for dancing	1	1
Index	13	6