1. Intro. This program makes DLX3 data for an interesting problem posed by Ian Cullis in 2022: Fill a 10×10 array with 1s, 2s, 3s, 4s so that there are exactly k occurrences of k in each row and each column. Also the 2s should form nontouching dominoes, the 3s should form nontouching trominoes, and the 4s should form nontouching ell-tetrominoes, where "nontouching" means not having edges in common.

This program is to be used with the UNIX command line

```
cat ian.dat | polyomino-dlx | ian-dlx
```

so that *stdin* contains appropriate data about the possible configurations of individual polynominoes and their boundaries.

```
#define bufsize 1024
#include <stdio.h>
#include <stdlib.h>
  char buf[bufsize];
  main()
  {
    register int i, j, k;
    ⟨Print the item-name line 2⟩;
    ⟨Print the options for individual cells 3⟩;
    ⟨Print the options for vetting polyominoes 4⟩;
  }
```

2. There are primary items R_{ik} and C_{jk} for $0 \le i, j < 10$ and $1 \le k \le 4$, indicating the number of ks in row or column k. There also are primary items $\#_{ij}$, meaning that cell ij has been "vetted" as a polyomino that matches its number.

There are secondary items ijk, which are essentially Boolean variables that state whether or not cell ij contains k.

I've also added primary items ij, with four options apiece. These aren't necessary, but they speed up the search.

```
⟨Print the item-name line 2⟩ ≡
  for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++) printf("%d%d␣", i, j);
  for (i = 0; i < 10; i++)
    for (k = 1; k ≤ 4; k++) printf("%d|R%d%d␣%d|C%d%d␣", k, i, k, k, i, k);
  for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++) printf("#%d%d␣", i, j);
  printf("|");
  for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
      for (k = 1; k ≤ 4; k++) printf("␣%d%d%d", i, j, k);
  printf("\n");
```

This code is used in section 1.

3. ⟨Print the options for individual cells 3⟩ ≡

```
for (i = 0; i < 10; i++)
  for (j = 0; j < 10; j++) {
    printf("%d%d␣R%d1␣C%d1␣%d%d1:1␣%d%d2:0␣%d%d3:0␣%d%d4:0\n", i, j, i, j, i, j, i, j, i, j, i, j);
    printf("%d%d␣R%d2␣C%d2␣%d%d1:0␣%d%d2:1␣%d%d3:0␣%d%d4:0\n", i, j, i, j, i, j, i, j, i, j, i, j);
    printf("%d%d␣R%d3␣C%d3␣%d%d1:0␣%d%d2:0␣%d%d3:1␣%d%d4:0\n", i, j, i, j, i, j, i, j, i, j, i, j);
    printf("%d%d␣R%d4␣C%d4␣%d%d1:0␣%d%d2:0␣%d%d3:0␣%d%d4:1\n", i, j, i, j, i, j, i, j, i, j, i, j);
  }
```

This code is used in section 1.

4. **#define** *less_one*(*k*) (*buf*[*k*] ≡ 'a' ? 9 : *buf*[*k*] − '1')

⟨ Print the options for vetting polyominoes 4 ⟩ ≡

```
while (1) {
  if (¬fgets(buf, bufsize, stdin)) break;
  switch (buf[0]) {
  case '|': case '␣': continue;
  case 'o': i = less_one(2), j = less_one(3);
    printf("#%d%d␣%d%d1:1", i, j, i, j);
    break;
  case 'd':
    for (k = 1; buf[k] ≡ '␣'; k += 3) {
      i = less_one(k + 1), j = less_one(k + 2);
      if (buf[k + 3] ≡ 'b') {
        k++;
        if (i ≥ 0 ∧ i < 10 ∧ j ≥ 0 ∧ j < 10) printf("%d%d2:0␣", i, j);
      } else {
        printf("#%d%d␣%d%d2:1␣", i, j, i, j);
      }
    }
    break;
  case 'v': case 't':
    for (k = 1; buf[k] ≡ '␣'; k += 3) {
      i = less_one(k + 1), j = less_one(k + 2);
      if (buf[k + 3] ≡ 'b') {
        k++;
        if (i ≥ 0 ∧ i < 10 ∧ j ≥ 0 ∧ j < 10) printf("%d%d3:0␣", i, j);
      } else {
        printf("#%d%d␣%d%d3:1␣", i, j, i, j);
      }
    }
    break;
  case 'l':
    for (k = 1; buf[k] ≡ '␣'; k += 3) {
      i = less_one(k + 1), j = less_one(k + 2);
      if (buf[k + 3] ≡ 'b') {
        k++;
        if (i ≥ 0 ∧ i < 10 ∧ j ≥ 0 ∧ j < 10) printf("%d%d4:0␣", i, j);
      } else {
        printf("#%d%d␣%d%d4:1␣", i, j, i, j);
      }
    }
    break;
  default: fprintf(stderr, "Bad␣input␣line!␣%s", buf);
  }
  printf("\n");
}
```

This code is used in section 1.

5. Index.

IAN-DLX