1. Intro. This program reads in a `.dots` file (see SAT-LIFE) and outputs clauses that will be satisfiable if and only if the 1s can be covered with dominoes having no three sharing a vertex. (Notice that I said 'no three', not 'no four'. This condition affects patterns with internal holes.)

The variables are *i*`H`*j* and *i*`V`*j*, meaning that pixel (i, j) is occupied by the left half of a horizontal domino or the top half of a vertical domino, respectively.

```
#define maxx 50     /* maximum number of lines in the pattern supplied by stdin */
#define maxy 200     /* maximum number of columns per line in stdin */
#include <stdio.h>
#include <stdlib.h>
  char p[maxx + 2][maxy + 2];     /* is cell (x, y) potentially alive? */
  int xmax, ymax;     /* the number of rows and columns in the input pattern */
  int xmin = maxx, ymin = maxy;     /* limits in the other direction */
  char buf[maxy + 2];     /* input buffer */
  char a[4][8];     /* place to assemble clauses */
  main()
  {
    register int i, j, k, x, y;
    ⟨ Input the pattern 2 ⟩;
    printf("~␣sat-tatami␣(%dx%d)\n", xmax, ymax);
    ⟨ Generate the clauses for domino covering 3 ⟩;
    ⟨ Generate the clauses to assert the tatami condition 4 ⟩;
  }
```

2. ⟨ Input the pattern 2 ⟩ ≡

```
  for (x = 1; ; x++) {
    if (¬fgets(buf, maxy + 2, stdin)) break;
    if (x > maxx) {
      fprintf(stderr, "Sorry,␣the␣pattern␣should␣have␣at␣most␣%d␣rows!\n", maxx);
      exit(−3);
    }
    for (y = 1; buf[y − 1] ≠ '\n'; y++) {
      if (y > maxy) {
        fprintf(stderr, "Sorry,␣the␣pattern␣should␣have␣at␣most␣%d␣columns!\n", maxy);
        exit(−4);
      }
      if (buf[y − 1] ≡ '*') {
        p[x][y] = 1;
        if (y > ymax) ymax = y;
        if (y < ymin) ymin = y;
        if (x > xmax) xmax = x;
        if (x < xmin) xmin = x;
      } else if (buf[y − 1] ≠ '.') {
        fprintf(stderr, "Unexpected␣character␣`%c'␣found␣in␣the␣pattern!\n", buf[y − 1]);
        exit(−5);
      }
    }
  }
```

This code is used in section 1.

3. Here I treat x as a row number and y as a column number. (Thus it's matrix notation, not Cartesian coordinates.)

```
⟨ Generate the clauses for domino covering 3 ⟩ ≡
  for (x = xmin; x ≤ xmax; x++)
    for (y = ymin; y ≤ ymax; y++)
      if (p[x][y]) {
        k = 0;
        if (p[x][y + 1]) sprintf(a[k], "%dH%d", x, y), k++;
        if (p[x][y − 1]) sprintf(a[k], "%dH%d", x, y − 1), k++;
        if (p[x + 1][y]) sprintf(a[k], "%dV%d", x, y), k++;
        if (p[x − 1][y]) sprintf(a[k], "%dV%d", x − 1, y), k++;
        if (k ≡ 0) {
          fprintf(stderr, "Cell␣(%d,", x);
          fprintf(stderr, "%d)␣cannot␣be␣covered␣with␣a␣domino!\n", y);
          exit(−1);
        }
        for (i = 0; i < k; i++)
          for (j = i + 1; j < k; j++) printf("~%s␣~%s\n", a[i], a[j]);      /* prevent overlap */
        for (i = 0; i < k; i++) printf("␣%s", a[i]);
        printf("\n");      /* force covering */
      }
```

This code is used in section 1.

4.
```
⟨ Generate the clauses to assert the tatami condition 4 ⟩ ≡
  for (x = xmin; x < xmax; x++)
    for (y = ymin; y < ymax; y++) {
      k = p[x][y] + p[x][y + 1] + p[x + 1][y] + p[x + 1][y + 1];
      if (k ≥ 3) {
        if (p[x][y] ∧ p[x][y + 1]) printf("␣%dH%d", x, y);
        if (p[x][y] ∧ p[x + 1][y]) printf("␣%dV%d", x, y);
        if (p[x + 1][y] ∧ p[x + 1][y + 1]) printf("␣%dH%d", x + 1, y);
        if (p[x][y + 1] ∧ p[x + 1][y + 1]) printf("␣%dV%d", x, y + 1);
        printf("\n");
      }
    }
```

This code is used in section 1.

5. Index.

SAT-TATAMI