1. Intro. Given the specification of a SuperSlitherlink puzzle in *stdin*, this program outputs MCC data for the problem of finding all solutions. (It's based on SLITHERLINK-DLX, which handles ordinary Slitherlink.)

SuperSlitherlink extends the former rules by allowing several cells to participate in the same clue. Each superclue is identified by a letter, and the relevant cells are marked with that letter. An edge that lies between two cells with the same letter is called "internal" and it cannot participate in the solution. An edge that lies between a cell with a letter and a cell that either has a numeric clue or a blank clue or lies off the board is a "boundary edge" for that letter. An edge that lies between cells with different letters is a boundary edge for both of those letters. The clue for a letter is the number of boundary edges that should appear in the solution.

No attempt is made to enforce the "single loop" condition. Solutions found by DLX3 will consist of disjoint loops. But DLX3-LOOP will weed out any disconnected solutions; in fact it will nip most of them in the bud.

The specification begins with m lines of n characters each; those characters should be either '`0`' or '`1`' or '`2`' or '`3`' or '`4`' or '`.`' or a letter. Those opening lines should be followed by lines of the form `!`⟨letter⟩`=`⟨clue⟩, one for each letter in the opening lines. For example, here's the specification for a simple SuperSlitherlink puzzle designed by Johan de Ruiter for Pi Day 2025:

```
aaa....
aaabb..
aaabb..
.......
..ccddd
..ccddd
....ddd
!a=3
!b=1
!c=4
!d=1
```

(See `https://cs.stanford.edu/~knuth/news25.html`.)

2. Here now is the general outline.

```
#define maxn 30    /* m and n must be at most this */
#define bufsize 80
#define panic(message)
        { fprintf(stderr, "%s:␣%s", message, buf); exit(−1); }
#include <stdio.h>
#include <stdlib.h>
  char buf[bufsize];
  int board[maxn][maxn];    /* the given clues */
  char super[128];    /* did this letter appear? */
  int clue[128];    /* numeric clues for letters that have appeared */
  char code[] = {#c, #a, #5, #3, #9, #6, #0};
  char edge[2 ∗ maxn + 1][2 ∗ maxn + 1];    /* is this a legal edge? */
  void main()
  {
    register int d, i, j, k, m, n;
    ⟨Read the input into board 3⟩;
    ⟨Determine the legal edges 5⟩;
    ⟨Print the item-name line 6⟩;
    for (i = 0; i ≤ m; i++)
      for (j = 0; j ≤ n; j++) {
        ⟨Print the options for tile (i, j) 7⟩;
        ⟨Print the options for subtile NW(i, j) 8⟩;
        ⟨Print the options for subtile NE(i, j) 9⟩;
        ⟨Print the options for subtile SE(i, j) 10⟩;
        ⟨Print the options for subtile SW(i, j) 11⟩;
        ⟨Print the options for subtile NS(i, j) 12⟩;
        ⟨Print the options for subtile EW(i, j) 13⟩;
      }
  }
```

3. ⟨Read the input into *board* 3⟩ ≡

```
printf("|␣slitherlink-dlx:\n");
for (i = 0; i < maxn; ) {
  if (¬fgets(buf, bufsize, stdin)) break;
  printf("|␣%s", buf);
  if (buf[0] ≡ '!') ⟨Record the clue for a letter and continue 4⟩;
  for (j = 0; j < maxn ∧ buf[j] ≠ '\n'; j++) {
    k = buf[j];
    if (k ≡ '|' ∨ k ≡ ':' ∨ k < 0) panic("Illegal␣character");
    if (k ≠ '.' ∧ (k < '0' ∨ k > '4')) super[k] = 1, clue[k] = −1;      /* we treat k as a letter */
    board[i][j] = k;
  }
  if (i++ ≡ 0) n = j;
  else if (n ≠ j) panic("row␣has␣wrong␣number␣of␣clues");
}
m = i;
if (m < 2 ∨ n < 2) panic("the␣board␣dimensions␣must␣be␣2␣or␣more");
for (k = 0; k < 128; k++)
  if (super[k] ∧ clue[k] < 0) fprintf(stderr, "No␣`!%c=...'!\n", k);
fprintf(stderr, "OK,␣I've␣read␣a␣%dx%d␣array␣of␣clues.\n", m, n);
```

This code is used in section 2.

4. ⟨Record the clue for a letter and **continue** 4⟩ ≡

```
{
  if (buf[2] ≠ '=') panic("no␣=␣sign");
  k = buf[1];
  if (¬super[k]) fprintf(stderr, "letter␣`%c'␣never␣occurred!\n", k);
  else {
    for (d = 0, j = 3; buf[j] ≥ '0' ∧ buf[j] ≤ '9'; j++) d = 10 * d + buf[j] − '0';
    clue[k] = d;
  }
  continue;
}
```

This code is used in section 3.

5. The primary items are "tiles," "cells," and "clues." Tiles control the loop path; the name of tile (i,j) is '$2i$**,**$2j$'. Cells represent numeric clues; the name of cell (i,j) is '$2i{+}1$**,**$2j{+}1$'. A cell item is present only if a numeric clue has been given for that cell of the board.

There also are primary items for "subtiles," which are somewhat subtle (please forgive the pun) and explained later. A subtile represents the state of two adjacent edges.

The secondary items are "edges" of the path. Their names are the midpoints of the tiles they connect.

An edge is illegal if it is internal to the clue of some letter. A boundary edge for a zero clue is also illegal.

```
⟨ Determine the legal edges 5 ⟩ ≡
  for ($i = 0$; $i \le m + m$; $i{+}{+}$)
    for ($j = 0$; $j \le n + n$; $j{+}{+}$)
      if (($i + j$) & 1) $edge[i][j] = 1$;
  for ($i = 0$; $i < m$; $i{+}{+}$)
    for ($j = 0$; $j < n$; $j{+}{+}$) {
      $k = board[i][j]$;
      if ($k \equiv$ '0')
        $edge[i+i][j+j+1] = edge[i+i+1][j+j] = edge[i+i+1][j+j+2] = edge[i+i+2][j+j+1] = 0$;
      else if ($super[k]$) {
        if ($j < n - 1 \land board[i][j+1] \equiv k$) $edge[i+i+1][j+j+2] = 0$;
        if ($i < m - 1 \land board[i+1][j] \equiv k$) $edge[i+i+2][j+j+1] = 0$;
        if ($clue[k] \equiv 0$) {
          $edge[i+i][j+j+1] = edge[i+i+1][j+j] = edge[i+i+1][j+j+2] = edge[i+i+2][j+j+1] = 0$;
          $super[k] = 0$;
        }
      }
    }
```

This code is used in section 2.

6. Each tile '$2i$,$2j$' controls the destiny of up to four edges that touch its central point (i,j). It has up to six subtiles, called '$2i$NW$2j$', '$2i$NE$2j$', '$2i$SE$2j$', '$2i$SW$2j$', '$2i$NS$2j$', '$2i$EW$2j$', which control the destinies of two edges at a time. A tile and its subtiles evolve together, in sync; the idea is to have a way to count the total number of chosen edges that belong to the boundary of each clue, without using the same clue name twice in any option.

For example, suppose $i = 1$ and $j = 2$. Tile '2,4' controls the destiny of the four edges that touch '2,4', namely '1,4', '3,4', '2,5', and '2,3' if we go north, south, east, and west from '2,4'. Its subtile 2NS4 controls just the first two of those four.

Each edge in the path appears in two tiles. So there will be c edges on a boundary of a region if and only if that region's name occurs in $2c$ options. (See '$2 * clue[k]$' in the following code.)

A two-coordinate "name" (x,y) actually appears as two encoded digits (in order to match the conventions of DLX3-LOOP).

```
#define encode(x)  ((x) < 10 ? (x) + '0' : (x) < 36 ? (x) − 10 + 'a' : (x) < 62 ? (x) − 36 + 'A' : '?')
#define N(i, j)  (i > 0 ∧ edge[i + i − 1][j + j])
#define W(i, j)  (j > 0 ∧ edge[i + i][j + j − 1])
#define E(i, j)  (j < n ∧ edge[i + i][j + j + 1])
#define S(i, j)  (i < m ∧ edge[i + i + 1][j + j])
```

```
⟨Print the item-name line 6⟩ ≡
  for (i = 0; i ≤ m; i++)
    for (j = 0; j ≤ n; j++) {
      printf("%c%c␣", encode(i + i), encode(j + j));
      if (N(i, j) ∧ W(i, j)) printf("%cNW%c␣", encode(i + i), encode(j + j));
      if (N(i, j) ∧ E(i, j)) printf("%cNE%c␣", encode(i + i), encode(j + j));
      if (S(i, j) ∧ E(i, j)) printf("%cSE%c␣", encode(i + i), encode(j + j));
      if (S(i, j) ∧ W(i, j)) printf("%cSW%c␣", encode(i + i), encode(j + j));
      if (N(i, j) ∧ S(i, j)) printf("%cNS%c␣", encode(i + i), encode(j + j));
      if (E(i, j) ∧ W(i, j)) printf("%cEW%c␣", encode(i + i), encode(j + j));
    }
  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
      if (board[i][j] ≥ '1' ∧ board[i][j] ≤ '4')
        printf("%d|%c%c␣", 2 * (board[i][j] − '0'), encode(i + i + 1), encode(j + j + 1));
  for (k = 0; k < 128; k++)
    if (super[k]) printf("%d|%c␣", 2 * clue[k], k);
  printf("|");
  for (i = 0; i ≤ m + m; i++)
    for (j = 0; j ≤ n + n; j++)
      if (edge[i][j]) printf("␣%c%c", encode(i), encode(j));
  printf("\n");
```

This code is used in section 2.

7. A tile is filled with either zero or two edges, and each edge potentially contributes to one or two clue counts. If there are two edges, we must be sure that they don't contribute to the same count.

Thus, with two edges, we might contribute to four counts. Two of them come from the tile; the other two from the corresponding subtile.

```
⟨Print the options for tile (i, j) 7⟩ ≡
  {
    for (k = 0; k < 7; k++) {
      if ((code[k] & 8) ∧ ¬N(i, j)) continue;    /* can't go north */
      if ((code[k] & 4) ∧ ¬W(i, j)) continue;    /* can't go west */
      if ((code[k] & 2) ∧ ¬E(i, j)) continue;    /* can't go east */
      if ((code[k] & 1) ∧ ¬S(i, j)) continue;    /* can't go south */
      printf("%c%c", encode(i + i), encode(j + j));
      if (N(i, j)) printf("␣%c%c:%d", encode(i + i − 1), encode(j + j), code[k] ≫ 3);
      if (W(i, j)) printf("␣%c%c:%d", encode(i + i), encode(j + j − 1), (code[k] ≫ 2) & 1);
      if (E(i, j)) printf("␣%c%c:%d", encode(i + i), encode(j + j + 1), (code[k] ≫ 1) & 1);
      if (S(i, j)) printf("␣%c%c:%d", encode(i + i + 1), encode(j + j), code[k] & 1);
      switch (k) {
      case 0: ⟨Show SW clue 17⟩; ⟨Show NW clue 14⟩; break;    /* NW */
      case 1: ⟨Show NW clue 14⟩; ⟨Show NE clue 15⟩; break;    /* NE */
      case 2: ⟨Show SE clue 16⟩; ⟨Show SW clue 17⟩; break;    /* SW */
      case 3: ⟨Show NE clue 15⟩; ⟨Show SE clue 16⟩; break;    /* SE */
      case 4: ⟨Show NW clue 14⟩; ⟨Show NE clue 15⟩; break;    /* NS */
      case 5: ⟨Show NE clue 15⟩; ⟨Show SE clue 16⟩; break;    /* EW */
      case 6: break;    /* untouched */
      }
      printf("\n");
    }
  }
```

This code is used in section 2.

8. The next six sections are almost the same, and rather boring. I hope I've got them right.

```
⟨Print the options for subtile NW(i, j) 8⟩ ≡
  if (N(i, j) ∧ W(i, j)) {
    printf("%cNW%c", encode(i + i), encode(j + j));
    if (N(i, j)) printf("␣%c%c:0", encode(i + i − 1), encode(j + j));
    if (W(i, j)) printf("␣%c%c:0", encode(i + i), encode(j + j − 1));
    printf("\n");
    if (N(i, j)) {
      printf("%cNW%c␣%c%c:1", encode(i + i), encode(j + j), encode(i + i − 1), encode(j + j));
      if (W(i, j)) printf("␣%c%c:0", encode(i + i), encode(j + j − 1));
      printf("\n");
    }
    if (W(i, j)) {
      printf("%cNW%c␣%c%c:1", encode(i + i), encode(j + j), encode(i + i), encode(j + j − 1));
      if (N(i, j)) printf("␣%c%c:0", encode(i + i − 1), encode(j + j));
      printf("\n");
    }
    if (N(i, j) ∧ W(i, j)) {
      printf("%cNW%c␣%c%c:1␣%c%c:1", encode(i + i), encode(j + j), encode(i + i − 1), encode(j + j),
          encode(i + i), encode(j + j − 1));
      ⟨Show NW clue 14⟩; ⟨Show NE clue 15⟩;
      printf("\n");
    }
  }
```

This code is used in section 2.

9.
```
⟨Print the options for subtile NE(i, j) 9⟩ ≡
  if (N(i, j) ∧ E(i, j)) {
    printf("%cNE%c", encode(i + i), encode(j + j));
    if (N(i, j)) printf("␣%c%c:0", encode(i + i − 1), encode(j + j));
    if (E(i, j)) printf("␣%c%c:0", encode(i + i), encode(j + j + 1));
    printf("\n");
    if (N(i, j)) {
      printf("%cNE%c␣%c%c:1", encode(i + i), encode(j + j), encode(i + i − 1), encode(j + j));
      if (E(i, j)) printf("␣%c%c:0", encode(i + i), encode(j + j + 1));
      printf("\n");
    }
    if (E(i, j)) {
      printf("%cNE%c␣%c%c:1", encode(i + i), encode(j + j), encode(i + i), encode(j + j + 1));
      if (N(i, j)) printf("␣%c%c:0", encode(i + i − 1), encode(j + j));
      printf("\n");
    }
    if (N(i, j) ∧ E(i, j)) {
      printf("%cNE%c␣%c%c:1␣%c%c:1", encode(i + i), encode(j + j), encode(i + i − 1), encode(j + j),
          encode(i + i), encode(j + j + 1));
      ⟨Show NE clue 15⟩; ⟨Show SE clue 16⟩;
      printf("\n");
    }
  }
```

This code is used in section 2.

10. ⟨ Print the options for subtile SE(i,j) 10 ⟩ ≡

```
if ($S(i,j) \land E(i,j)$) {
  printf("%cSE%c", encode($i+i$), encode($j+j$));
  if ($S(i,j)$) printf("␣%c%c:0", encode($i+i+1$), encode($j+j$));
  if ($E(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j+1$));
  printf("\n");
  if ($S(i,j)$) {
    printf("%cSE%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i+1$), encode($j+j$));
    if ($E(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j+1$));
    printf("\n");
  }
  if ($E(i,j)$) {
    printf("%cSE%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i$), encode($j+j+1$));
    if ($S(i,j)$) printf("␣%c%c:0", encode($i+i+1$), encode($j+j$));
    printf("\n");
  }
  if ($S(i,j) \land E(i,j)$) {
    printf("%cSE%c␣%c%c:1␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i+1$), encode($j+j$),
        encode($i+i$), encode($j+j+1$));
    ⟨ Show SE clue 16 ⟩; ⟨ Show SW clue 17 ⟩;
    printf("\n");
  }
}
```

This code is used in section 2.

11. ⟨ Print the options for subtile SW(i,j) 11 ⟩ ≡

```
if ($S(i,j) \land W(i,j)$) {
  printf("%cSW%c", encode($i+i$), encode($j+j$));
  if ($S(i,j)$) printf("␣%c%c:0", encode($i+i+1$), encode($j+j$));
  if ($W(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j-1$));
  printf("\n");
  if ($S(i,j)$) {
    printf("%cSW%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i+1$), encode($j+j$));
    if ($W(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j-1$));
    printf("\n");
  }
  if ($W(i,j)$) {
    printf("%cSW%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i$), encode($j+j-1$));
    if ($S(i,j)$) printf("␣%c%c:0", encode($i+i+1$), encode($j+j$));
    printf("\n");
  }
  if ($S(i,j) \land W(i,j)$) {
    printf("%cSW%c␣%c%c:1␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i+1$), encode($j+j$),
        encode($i+i$), encode($j+j-1$));
    ⟨ Show SW clue 17 ⟩; ⟨ Show NW clue 14 ⟩;
    printf("\n");
  }
}
```

This code is used in section 2.

12. ⟨ Print the options for subtile NS(i,j) 12 ⟩ ≡

```
if ($N(i,j) \land S(i,j)$) {
  printf("%cNS%c", encode($i+i$), encode($j+j$));
  if ($N(i,j)$) printf("␣%c%c:0", encode($i+i-1$), encode($j+j$));
  if ($S(i,j)$) printf("␣%c%c:0", encode($i+i+1$), encode($j+j$));
  printf("\n");
  if ($N(i,j)$) {
    printf("%cNS%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i-1$), encode($j+j$));
    if ($S(i,j)$) printf("␣%c%c:0", encode($i+i+1$), encode($j+j$));
    printf("\n");
  }
  if ($S(i,j)$) {
    printf("%cNS%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i+1$), encode($j+j$));
    if ($N(i,j)$) printf("␣%c%c:0", encode($i+i-1$), encode($j+j$));
    printf("\n");
  }
  if ($N(i,j) \land S(i,j)$) {
    printf("%cNS%c␣%c%c:1␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i-1$), encode($j+j$),
        encode($i+i+1$), encode($j+j$));
    ⟨ Show SE clue 16 ⟩; ⟨ Show SW clue 17 ⟩;
    printf("\n");
  }
}
```

This code is used in section 2.

13. ⟨ Print the options for subtile EW(i,j) 13 ⟩ ≡

```
if ($E(i,j) \land W(i,j)$) {
  printf("%cEW%c", encode($i+i$), encode($j+j$));
  if ($E(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j+1$));
  if ($W(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j-1$));
  printf("\n");
  if ($E(i,j)$) {
    printf("%cEW%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i$), encode($j+j+1$));
    if ($W(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j-1$));
    printf("\n");
  }
  if ($W(i,j)$) {
    printf("%cEW%c␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i$), encode($j+j-1$));
    if ($E(i,j)$) printf("␣%c%c:0", encode($i+i$), encode($j+j+1$));
    printf("\n");
  }
  if ($E(i,j) \land W(i,j)$) {
    printf("%cEW%c␣%c%c:1␣%c%c:1", encode($i+i$), encode($j+j$), encode($i+i$), encode($j+j+1$),
        encode($i+i$), encode($j+j-1$));
    ⟨ Show SW clue 17 ⟩; ⟨ Show NW clue 14 ⟩;
    printf("\n");
  }
}
```

This code is used in section 2.

14. Finally we need to identify the clues, if any, that are relevant in each of the four quadrants of tile (i, j).

```
⟨Show NW clue 14⟩ ≡
  if $(i > 0 \land j > 0)$ {
    register int $k = board[i-1][j-1]$;
    if $(k \neq$ '.'$)$ {
      if $(k \geq$ '1' $\land k \leq$ '4'$)$ printf("␣%c%c", encode(i + i − 1), encode(j + j − 1));
      else printf("␣%c", k);
    }
  }
```

This code is used in sections 7, 8, 11, and 13.

15.

```
⟨Show NE clue 15⟩ ≡
  if $(i > 0 \land j < n)$ {
    register int $k = board[i-1][j]$;
    if $(k \neq$ '.'$)$ {
      if $(k \geq$ '1' $\land k \leq$ '4'$)$ printf("␣%c%c", encode(i + i − 1), encode(j + j + 1));
      else printf("␣%c", k);
    }
  }
```

This code is used in sections 7, 8, and 9.

16.

```
⟨Show SE clue 16⟩ ≡
  if $(i < m \land j < n)$ {
    register int $k = board[i][j]$;
    if $(k \neq$ '.'$)$ {
      if $(k \geq$ '1' $\land k \leq$ '4'$)$ printf("␣%c%c", encode(i + i + 1), encode(j + j + 1));
      else printf("␣%c", k);
    }
  }
```

This code is used in sections 7, 9, 10, and 12.

17.

```
⟨Show SW clue 17⟩ ≡
  if $(i < m \land j > 0)$ {
    register int $k = board[i][j-1]$;
    if $(k \neq$ '.'$)$ {
      if $(k \geq$ '1' $\land k \leq$ '4'$)$ printf("␣%c%c", encode(i + i + 1), encode(j + j − 1));
      else printf("␣%c", k);
    }
  }
```

This code is used in sections 7, 10, 11, 12, and 13.

18. Index.

board: 2, 3, 5, 6, 14, 15, 16, 17.
buf: 2, 3, 4.
bufsize: 2, 3.
clue: 2, 3, 4, 5, 6.
code: 2, 7.
d: 2.
E: 6.
edge: 2, 5, 6.
encode: 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17.
exit: 2.
fgets: 3.
fprintf: 2, 3, 4.
i: 2.
j: 2.
k: 2, 14, 15, 16, 17.
m: 2.
main: 2.
maxn: 2, 3.
message: 2.
N: 6.
n: 2.
panic: 2, 3, 4.
printf: 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17.
S: 6.
stderr: 2, 3, 4.
stdin: 1, 3.
super: 2, 3, 4, 5, 6.
W: 6.

$\langle$ Determine the legal edges 5 $\rangle$ Used in section 2.
$\langle$ Print the item-name line 6 $\rangle$ Used in section 2.
$\langle$ Print the options for subtile EW(i,j) 13 $\rangle$ Used in section 2.
$\langle$ Print the options for subtile NE(i,j) 9 $\rangle$ Used in section 2.
$\langle$ Print the options for subtile NS(i,j) 12 $\rangle$ Used in section 2.
$\langle$ Print the options for subtile NW(i,j) 8 $\rangle$ Used in section 2.
$\langle$ Print the options for subtile SE(i,j) 10 $\rangle$ Used in section 2.
$\langle$ Print the options for subtile SW(i,j) 11 $\rangle$ Used in section 2.
$\langle$ Print the options for tile (i,j) 7 $\rangle$ Used in section 2.
$\langle$ Read the input into *board* 3 $\rangle$ Used in section 2.
$\langle$ Record the clue for a letter and **continue** 4 $\rangle$ Used in section 3.
$\langle$ Show NE clue 15 $\rangle$ Used in sections 7, 8, and 9.
$\langle$ Show NW clue 14 $\rangle$ Used in sections 7, 8, 11, and 13.
$\langle$ Show SE clue 16 $\rangle$ Used in sections 7, 9, 10, and 12.
$\langle$ Show SW clue 17 $\rangle$ Used in sections 7, 10, 11, 12, and 13.

SUPERSLITHERLINK-GENERAL-DLX

	Section	Page
Intro	1	1
Index	18	11