1. Intro. Given the specification of a slitherlink puzzle in *stdin*, this program outputs DLX data for the problem of finding all solutions.

No attempt is made to enforce the “single loop” condition. Solutions found by DLX2 will consist of disjoint loops. But DLX2-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 ‘`.`’.

```
#define maxn 30     /* m and n must be at most this */
#define bufsize 80
#define debug 1     /* verbose printing? */
#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 code[ ] = {#c, #a, #5, #3, #9, #6, #0};
  char bin[ ] = {#0, #1, #2, #4, #8, #3, #5, #6, #9, #a, #c, #7, #b, #d, #e, #f};
  char start[ ] = {0, 1, 5, 11, 15, 16};
  main( )
  {
    register int i, j, k, m, n;
    ⟨Read the input into board 2⟩;
    ⟨Print the item-name line 3⟩;
    for (i = 0; i ≤ m; i++)
      for (j = 0; j ≤ n; j++) ⟨Print the options for tile (i, j) 4⟩;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        if (board[i][j] ≠ '.') ⟨Print the options for cell (i, j) 5⟩;
  }
```

2. ⟨Read the input into *board* 2⟩ ≡

```
  printf("|␣slitherlink-dlx:\n");
  for (i = 0; i < maxn; i++) {
    if (¬fgets(buf, bufsize, stdin)) break;
    printf("|␣%s", buf);
    for (j = 0; j < maxn ∧ buf[j] ≠ '\n'; j++) {
      k = buf[j];
      if (k ≠ '.' ∧ (k < '0' ∨ k > '4')) panic("illegal␣clue");
      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");
  fprintf(stderr, "OK,␣I've␣read␣a␣%dx%d␣array␣of␣clues.\n", m, n);
```

This code is used in section 1.

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

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

```
#define encode(x) ((x) < 10 ? (x) + '0' : (x) < 36 ? (x) − 10 + 'a' : (x) < 62 ? (x) − 36 + 'A' : '?')
```

⟨ Print the item-name line 3 ⟩ ≡

```
for (i = 0; i ≤ m; i++)
  for (j = 0; j ≤ n; j++) printf("%c%c␣", encode(i + i), encode(j + j));
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++)
    if (board[i][j] ≠ '.') printf("%c%c␣", encode(i + i + 1), encode(j + j + 1));
printf("|");
for (i = 0; i ≤ m + m; i++)
  for (j = 0; j ≤ n + n; j++)
    if ((i + j) & 1) printf("␣%c%c", encode(i), encode(j));
printf("\n");
```

This code is used in section 1.

4.

```
#define N(ii, jj) encode(ii − 1), encode(jj)
#define S(ii, jj) encode(ii + 1), encode(jj)
#define E(ii, jj) encode(ii), encode(jj + 1)
#define W(ii, jj) encode(ii), encode(jj − 1)
```

⟨ Print the options for tile (i,j) 4 ⟩ ≡

```
{
  for (k = 0; k < 7; k++) {
    if (i ≡ 0 ∧ (code[k] & 8)) continue;   /* can't go north */
    if (j ≡ 0 ∧ (code[k] & 4)) continue;   /* can't go west */
    if (j ≡ n ∧ (code[k] & 2)) continue;   /* can't go east */
    if (i ≡ m ∧ (code[k] & 1)) continue;   /* can't go south */
    printf("%c%c", encode(i + i), encode(j + j));
    if (i > 0) printf("␣%c%c:%d", N(i + i, j + j), code[k] ≫ 3);
    if (j > 0) printf("␣%c%c:%d", W(i + i, j + j), (code[k] ≫ 2) & 1);
    if (j < n) printf("␣%c%c:%d", E(i + i, j + j), (code[k] ≫ 1) & 1);
    if (i < m) printf("␣%c%c:%d", S(i + i, j + j), code[k] & 1);
    printf("\n");
  }
}
```

This code is used in section 1.

5. ⟨ Print the options for cell (i,j) 5 ⟩ ≡

```
{
  for (k = start[board[i][j] − '0']; k < start[board[i][j] − '0' + 1]; k++)
    printf("%c%c␣%c%c:%d␣%c%c:%d␣%c%c:%d␣%c%c:%d\n", encode(i + i + 1), encode(j + j + 1),
        N(i + i + 1, j + j + 1), bin[k] ≫ 3, W(i + i + 1, j + j + 1), (bin[k] ≫ 2) & 1, E(i + i + 1, j + j + 1),
        (bin[k] ≫ 1) & 1, S(i + i + 1, j + j + 1), bin[k] & 1);
}
```

This code is used in section 1.

6. Index.

⟨ Print the item-name line 3 ⟩ Used in section 1.
⟨ Print the options for cell (i, j) 5 ⟩ Used in section 1.
⟨ Print the options for tile (i, j) 4 ⟩ Used in section 1.
⟨ Read the input into *board* 2 ⟩ Used in section 1.

SLITHERLINK-DLX