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

What is a generalized sudoku puzzle? It is a puzzle whose specification begins with n lines of n characters each, where n is between 1 and 32. The characters on these lines are of three kinds:

- A digit from 1 to n. (Those digits are `1`, `2`, ..., `9`, `a`, `b`, ..., `w`.) This means that the puzzle will contain this digit as a clue in this cell.
- The character '`#`'. This means that this cell is a "hole" in the puzzle, not meant to be filled in.
- Any other character. This means that this cell is initially blank.

The specification continues with zero or more additional groups of n lines of n characters. These groups specify "boxes" (also called "regions"). The characters on these lines are of two kinds:

- A digit from 0 to $n-1$. (Those digits are `0`, `1`, ..., `9`, `a`, `b`, ..., `v`.) This means that the cell is part of the box that has this name.
- The character '`.`'. This means nothing. I mean, it means that nothing about boxes is being specified for this cell in this group.

Boxes can overlap, but only if they're specified in different groups. When the input has ended, every box that has been specified should contain at most n cells.

What is the solution to a generalized sudoku puzzle? It is a way to fill in all of the initially blank cells, with digits from 1 to n, in such a way that no digit occurs more than once in any row, column, or box.

Here, for example, is the letter 'A' from the Puzzlium ABC, which was presented by Serhiy and Peter Grabarchuk at the Martin Gardner Centennial celebration in Berkeley on 26 October 2014:

```
#.5..#
.4..3.
6.##..
.5.2..
.....1
.3##..
.0000.
122203
12..03
122433
144443
11..43
```

It specifies five hexomino boxes. (The reader will enjoy finding its solution.)

The clues are repeated in a comment line at the beginning of the output.

```
#define bufsize 80
#include <stdio.h>
#include <stdlib.h>
  char buf[bufsize];
  int pos[32][32];      /* clues and holes */
  int row[32][32];      /* does this row contain this clue? */
  int col[32][32];      /* does this column contain this clue? */
  int box[32][32];      /* does this box contain this clue? */
  int rowcount[32], colcount[32], boxcount[32];      /* how many cells in this guy? */
  int c;      /* how many clues have been given? */
  int bc;      /* how many boxes have been defined? */
  int cells;      /* how many cells are left, after holes deducted? */
  unsigned int inbox[32][32];      /* which boxes contain this cell? */
  main()
  {
```

```
  register int d, i, j, k, kk, n, x;
  ⟨ Input the given problem 2 ⟩;
  ⟨ Output the comment line 6 ⟩;
  ⟨ Output the item-name line 7 ⟩;
  ⟨ Output the options 8 ⟩;
}
```

2. ⟨ Input the given problem 2 ⟩ ≡

```
for (n = k = kk = 0; ; kk++) {
  if (¬fgets(buf, bufsize, stdin)) break;
  ⟨ Make sure buf has exactly n characters 3 ⟩;
  if (kk < n) ⟨ Input line k of the overall spec 4 ⟩
  else ⟨ Input line k of a box-definition group 5 ⟩;
}
if (kk < n) {
  fprintf(stderr, "There were fewer than %d lines of input!\n", n);
  exit(−5);
}
;
if (k + 1 < n) {
  fprintf(stderr, "Box-definition group %d had fewer than %d lines of input!\n", kk/n, n);
  exit(−6);
}
fprintf(stderr, "OK, I've got n=%d, with %d boxes and %d clues in %d cells.\n", n, bc, c, cells);
```

This code is used in section 1.

3. ⟨ Make sure *buf* has exactly n characters 3 ⟩ ≡

```
if (¬n) {      /* this is the first line, which has n chars by definition */
  for (n = 0; buf[n] ∧ buf[n] ≠ '\n'; n++) ;      /* advance to end of line */
  if (n ≡ 0) {
    fprintf(stderr, "the␣length␣of␣the␣first␣line␣(n)␣is␣zero!\n");
    exit(−1);
  }
  if (n > 32) {
    fprintf(stderr, "the␣length␣of␣the␣first␣line␣(%d)␣exceeds␣32!\n", n);
    exit(−2);
  }
  cells = n ∗ n;
  for (j = 0; j < n; j++) rowcount[j] = colcount[j] = n;
}
else {
  k = kk % n;
  for (j = 0; j < n; j++)
    if (buf[j] ≡ '\n') {
      fprintf(stderr, "input␣line␣%d␣has␣fewer␣than␣%d␣characters!\n", kk, n);
      exit(−3);
    }
  if (buf[j] ≠ '\n') {
    fprintf(stderr, "input␣line␣%d␣has␣more␣than␣%d␣characters!\n", kk, n);
    exit(−4);
  }
}
```

This code is used in section 2.

4. **#define** *encode*(*d*) ((*d*) < 10 ? '0' + (*d*) : 'a' + (*d*) − 10)

⟨ Input line k of the overall spec 4 ⟩ ≡

```
for (j = 0; j < n; j++) {
  if (buf[j] > '0' ∧ buf[j] ≤ '9') pos[k][j] = d = buf[j] − '0';
  else if (buf[j] ≥ 'a' ∧ buf[j] ≤ 'w') pos[k][j] = d = buf[j] − 'a' + 10;
  else if (buf[j] ≡ '#') pos[k][j] = −1, cells−−, rowcount[k]−−, colcount[j]−−;
  else pos[k][j] = 0;      /* it already is zero, but let's waste time for clarity */
  if (pos[k][j] > 0) {
    if (row[k][d − 1]) {
      fprintf(stderr, "digit␣%c␣appears␣in␣columns␣%c␣and␣%c␣of␣row␣%c!\n", encode(d),
          encode(row[k][d − 1] − 1), encode(j), encode(k));
      exit(−10);
    }
    row[k][d − 1] = j + 1;
    if (col[j][d − 1]) {
      fprintf(stderr, "digit␣%c␣appears␣in␣rows␣%c␣and␣%c␣of␣column␣%c!\n", encode(d),
          encode(col[j][d − 1] − 1), encode(k), encode(j));
      exit(−11);
    }
    col[j][d − 1] = k + 1;
    c++;
  }
}
```

This code is used in section 2.

5. ⟨ Input line k of a box-definition group 5 ⟩ ≡

```
for (j = 0; j < n; j++) {
  if (buf[j] ≡ '.') continue;
  if (buf[j] ≥ '0' ∧ buf[j] ≤ '9') x = buf[j] − '0';
  else if (buf[j] ≥ 'a' ∧ buf[j] ≤ 'v') x = buf[j] − 'a' + 10;
  else {
    fprintf(stderr, "line␣%d␣of␣box-definition␣group␣%d␣has␣the␣invalid␣character␣%c!\n", k,
        kk/n, buf[j]);
    exit(−7);
  }
  d = pos[k][j];
  if (d > 0) {
    if (box[x][d − 1]) {
      fprintf(stderr, "digit␣%c␣appears␣in␣rows␣%c␣and␣%c␣of␣box␣%c!\n", encode(d),
          encode(box[x][d − 1] − 1), encode(k), encode(x));
      exit(−12);
    }
    box[x][d − 1] = k + 1;
  }
  if (boxcount[x] ≡ 0) bc++;
  if (inbox[k][j] & (1 ≪ x)) {
    fprintf(stderr, "box␣%c␣already␣contains␣the␣cell␣in␣row␣%c,␣column␣%c!\n", encode(x),
        encode(k), encode(j));
    exit(−13);
  }
  inbox[k][j] |= 1 ≪ x, boxcount[x]++;
  if (boxcount[x] > n) {
    fprintf(stderr, "box␣%c␣contains␣more␣than␣%d␣cells!\n", encode(x), n);
    exit(−13);
  }
}
```

This code is used in section 2.

6. ⟨ Output the comment line 6 ⟩ ≡

```
printf("|sudoku");
for (i = 0; i < n; i++) {
  printf("!");
  for (j = 0; j < n; j++)
    fprintf(stdout, "%c", pos[i][j] < 0 ? '#' : pos[i][j] > 0 ? encode(pos[i][j]) : '.');
}
fprintf(stdout, "\n");
```

This code is used in section 1.

7. The `p` items precede the `r` items, which precede the `c` items, which precede the `b` items. An item is omitted if there already was a clue for it. An item is secondary if it doesn't need to appear n times.

```
⟨ Output the item-name line 7 ⟩ ≡
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (pos[i][j] ≡ 0) printf("p%c%c␣", encode(i), encode(j));
  for (i = 0; i < n; i++)
    for (d = 0; d < n; d++)
      if (rowcount[i] ≡ n ∧ ¬row[i][d]) printf("r%c%c␣", encode(i), encode(d + 1));
  for (j = 0; j < n; j++)
    for (d = 0; d < n; d++)
      if (colcount[j] ≡ n ∧ ¬col[j][d]) printf("c%c%c␣", encode(j), encode(d + 1));
  for (x = 0; x < 32; x++)
    for (d = 0; d < n; d++)
      if (boxcount[x] ≡ n ∧ ¬box[x][d]) printf("b%c%c␣", encode(x), encode(d + 1));
  printf("|");
  for (i = 0; i < n; i++)
    for (d = 0; d < n; d++)
      if (rowcount[i] ∧ rowcount[i] < n ∧ ¬row[i][d]) printf("␣r%c%c", encode(i), encode(d + 1));
  for (j = 0; j < n; j++)
    for (d = 0; d < n; d++)
      if (colcount[j] ∧ colcount[j] < n ∧ ¬col[j][d]) printf("␣c%c%c", encode(j), encode(d + 1));
  for (x = 0; x < 32; x++)
    for (d = 0; d < n; d++)
      if (boxcount[x] ∧ boxcount[x] < n ∧ ¬box[x][d]) printf("␣b%c%c", encode(x), encode(d + 1));
  printf("\n");
```

This code is used in section 1.

8.

```
⟨ Output the options 8 ⟩ ≡
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      for (d = 0; d < n; d++) {
        if (pos[i][j] ≠ 0 ∨ row[i][d] ≠ 0 ∨ col[j][d] ≠ 0) continue;
        for (x = 0; x < 32; x++) {
          if ((inbox[i][j] & (1 ≪ x)) ≡ 0) continue;
          if (box[x][d] ≠ 0) break;
        }
        if (x < 32) continue;
        printf("p%c%c␣r%c%c␣c%c%c", encode(i), encode(j), encode(i), encode(d + 1), encode(j),
            encode(d + 1));
        for (x = 0; x < 32; x++) {
          if ((inbox[i][j] & (1 ≪ x)) ≡ 0) continue;
          printf("␣b%c%c", encode(x), encode(d + 1));
        }
        printf("\n");
      }
```

This code is used in section 1.

9. Index.

⟨ Input line k of a box-definition group 5 ⟩ Used in section 2.
⟨ Input line k of the overall spec 4 ⟩ Used in section 2.
⟨ Input the given problem 2 ⟩ Used in section 1.
⟨ Make sure *buf* has exactly n characters 3 ⟩ Used in section 2.
⟨ Output the comment line 6 ⟩ Used in section 1.
⟨ Output the item-name line 7 ⟩ Used in section 1.
⟨ Output the options 8 ⟩ Used in section 1.

SUDOKU-GENERAL-DLX