1. Intro. This program generates DLX3 data that finds all "motley dissections" of an $m \times n$ rectangle into subrectangles.

The allowable subrectangles $[a\,.\,.\,b) \times [c\,.\,.\,d)$ have $0 \le a < b \le m$, $0 \le c < d \le n$, with $(a,b) \ne (0,m)$ and $(c,d) \ne (0,n)$; so there are $\bigl(\binom{m+1}{2} - 1\bigr) \cdot \bigl(\binom{n+1}{2} - 1\bigr)$ possibilities. Such a dissection is *motley* if the pairs (a,b) are distinct, and so are the pairs (c,d); in other words, no two subrectangles have identical top-bottom boundaries or left-right boundaries.

Furthermore we require that every $x \in [0\,.\,.\,m)$ occurs at least once among the a's; every $y \in [0\,.\,.\,n)$ occurs at least once among the c's. Otherwise the dissection could be collapsed into a smaller one, by leaving out that coordinate value.

It turns out that we can save a factor of (roughly) 2 by using symmetry, and looking at the unique rectangles that lie within the top and bottom rows of every solution.

```
#define maxd 36   /* maximum value for m or n */
#define encode(v) ((v) < 10 ? (v) + '0' : (v) - 10 + 'a')   /* encoding for values < 36 */
#include <stdio.h>
#include <stdlib.h>
  int m, n;   /* command-line parameters */
  main(int argc, char *argv[])
  {
    register int a, b, c, d, j, k;
    ⟨Process the command line 2⟩;
    ⟨Output the first line 3⟩;
    for (a = 0; a < m; a++)
      for (b = a + 1; b ≤ m; b++)
        if (a ≠ 0 ∨ b ≠ m) {
          for (c = 0; c < n; c++)
            for (d = c + 1; d ≤ n; d++)
              if (c ≠ 0 ∨ d ≠ n) {⟨Output the line for [a..b] × [c..d] 5⟩}
        }
  }
```

2. ⟨Process the command line 2⟩ ≡

```
  if (argc ≠ 3 ∨ sscanf(argv[1], "%d", &m) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1) {
    fprintf(stderr, "Usage: %s m n\n", argv[0]);
    exit(−1);
  }
  if (m > maxd ∨ n > maxd) {
    fprintf(stderr, "Sorry, m and n must be at most %d!\n", maxd);
    exit(−2);
  }
  printf("| motley-dlx %d %d\n", m, n);
```

This code is used in section 1.

3. The main primary columns jk ensure that cell (j,k) is covered, for $0 \le j < m$ and $0 \le k < n$. We also have secondary columns xab and ycd, to ensure that no interval is repeated. And there are primary columns xa and yc for the at-least-once conditions.

⟨ Output the first line 3 ⟩ ≡

```
for (j = 0; j < m; j++)
  for (k = 0; k < n; k++) printf("␣%c%c", encode(j), encode(k));
for (a = 1; a < m; a++) printf("␣1:%d|x%c", m − a, encode(a));
for (c = 1; c < n; c++) printf("␣1:%d|y%c", n − c, encode(c));
printf("␣|");
for (a = 0; a < m; a++)
  for (b = a + 1; b ≤ m; b++)
    if (a ≠ 0 ∨ b ≠ m) printf("␣x%c%c", encode(a), encode(b));
for (c = 0; c < n; c++)
  for (d = c + 1; d ≤ n; d++)
    if (c ≠ 0 ∨ d ≠ n) printf("␣y%c%c", encode(c), encode(d));
⟨ Output also the secondary columns for symmetry breaking 6 ⟩;
printf("\n");
```

This code is used in section 1.

4. Now let's look closely at the problem of breaking symmetry. For concreteness, let's suppose that $m = 7$ and $n = 8$. Every solution will have exactly one entry with interval x67, specifying a rectangle in the bottom row (since $m - 1 = 6$). If that rectangle has y57, say, a left-right reflection would produce an equivalent solution with y13; therefore we do not allow the rectangle for which $(a, b, c, d) = (6, 7, 5, 7)$. Similarly we disallow $(6, 7, c, d)$ whenever $8 - d < c$, since we'll find all solutions with $(6, 7, 8 - d, 8 - c)$ that are left-right reflections of the solutions excluded.

If $a = 6$, $b = 7$, and $c + d = 8$, left-right reflection doesn't affect the rectangle in the bottom row. But we can still break the symmetry by looking at the top row, the rectangle whose specifications (a', b', c', d') have $(a', b') = (0, 1)$. Let's introduce secondary columns !1, !2, !3, using !c when $c+d = 8$ at the bottom. Then if we put !1, !2, and !3 on every top-row rectangle with $c' + d' > 8$, we'll forbid such rectangles whenever the bottom-row policy has not already broken left-right symmetry. Furthermore, when $c' + d' = 8$ at the top, we put !1 together with x01 y26, and we put both !1 and !2 together with x01 y35. It can be seen that this completely breaks left-symmetry in all cases, because no solution has $c = c'$ and $d = d'$.

(Think about it.)

It's tempting to believe, as the author once did, that the same idea will break top-bottom symmetry too. But that would be fallacious: Once we've fixed attention on the bottommost row while breaking left-right symmetry, we no longer have any symmetry between top and bottom.

(Think about that, too.)

5. ⟨Output the line for $[a \,..\, b] \times [c \,..\, d]$ 5⟩ ≡

```
if (a ≡ m − 1 ∧ c + d > n) continue;     /* forbid this case */
for (j = a; j < b; j++)
  for (k = c; k < d; k++) printf("␣%c%c", encode(j), encode(k));
if (a ≡ m − 1 ∧ c + d ≡ n) printf("␣!%d", c);     /* flag a symmetric bottom row */
if (b ≡ 1 ∧ c + d ≥ n) {     /* disallow top rectangle if bottom one is symmetric */
  if (c + d ≠ n)
    for (k = 1; k + k < n; k++) printf("␣!%d", k);
  else
    for (k = 1; k < c; k++) printf("␣!%d", k);
}
if (a) printf("␣x%c", encode(a));
if (c) printf("␣y%c", encode(c));
printf("␣x%c%c␣y%c%c\n", encode(a), encode(b), encode(c), encode(d));
```

This code is used in section 1.

6. ⟨Output also the secondary columns for symmetry breaking 6⟩ ≡

```
for (k = 1; k + k < n; k++) printf("␣!%d", k);
```

This code is used in section 3.

7. Index.

MOTLEY-DLX