1. Intro. This program generates all Kepler towers made from n bricks. (It supplements the old program VIENNOT in my Mittag-Leffler report “Three Catalan bijections,” which was incomplete: The claim that all towers are generated was never proved, because I'd blithely assumed that there are no more than C_n of them.)

```
#define maxn 40     /* this is plenty big, since C_40 > 10^21 */
#include <stdio.h>
#include <stdlib.h>
  int n;     /* command line parameter */
  int x[maxn];     /* current brick position */
  int w[maxn];     /* current wall number */
  int p[maxn];     /* beginning of supporting layer */
  int q[maxn];     /* beginning of current layer */
  int t[maxn];     /* type of move: 1 if end of layer, 2 if end of wall */
  char punct[3] = {',', ';', ':'};     /* separators */
  unsigned long long count;     /* this many found */
  main(int argc, char *argv[])
  {
    register i, j, k, l, mask;
    ⟨Process the command line 2⟩;
  b1: ⟨Initialize for backtracking 4⟩;
  b2: if (l > n) ⟨Visit a solution and goto b5 3⟩;
    w[l] = w[l − 1];
    switch (t[l − 1]) {
    case 0: x[l] = x[l − 1] + 2;     /* add brick to the current layer */
      if (x[l] > (1 ≪ w[l])) goto b5;     /* oops, it's out of range */
      break;
    case 2: fprintf(stderr, "This␣can't␣happen.\n");
    case 1: x[l] = 1; break;
    }
  b3: ⟨Test if a brick at x[l] is supported; if so, add it 5⟩;
  b4: ⟨Advance to the next trial move 6⟩;
  b5: if (−−l) {
      if (p[l] ≡ 0 ∧ t[l] ≠ 1) goto b5;     /* we're backtracking to previous wall */
      goto b4;
    }
    fprintf(stderr, "Altogether␣%lld␣towers␣generated.\n", count);
  }
```

2. ⟨Process the command line 2⟩ ≡

```
  if (argc ≠ 2 ∨ sscanf(argv[1], "%d", &n) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣n\n", argv[0]);
    exit(−1);
  }
  if (n > maxn) {
    fprintf(stderr, "You␣must␣be␣kidding;␣I␣can't␣handle␣n>%d!\n", maxn);
    exit(−2);
  }
```

This code is used in section 1.

3. ⟨Visit a solution and **goto** *b5* 3⟩ ≡

```
{
  count++;
  if (n ≤ 10)
    for (j = 1; j ≤ n; j++) printf("%d%c", x[j], j < n ? punct[t[j]] : '\n');
  t[l − 1] = 1;     /* complete the top layer */
  goto b5;
}
```

This code is used in section 1.

4. ⟨Initialize for backtracking 4⟩ ≡

```
l = 0, t[0] = 1;
goto b4;
```

This code is used in section 1.

5. ⟨Test if a brick at $x[l]$ is supported; if so, add it 5⟩ ≡

```
if (t[l − 1]) q[l] = l, p[l] = q[l − 1];
else q[l] = q[l − 1], p[l] = p[l − 1];
if (x[l] ≡ (1 ≪ w[l]) ∧ x[q[l]] ≡ 1) goto b5;     /* clashing bricks in ring */
mask = (1 ≪ w[l]) − 1;
for (j = p[l]; j < q[l]; j++)
  if (((x[j] ⊕ (x[l] − 1)) & mask) ≡ 0 ∨ x[j] ≡ x[l] ∨ ((x[j] ⊕ (x[l] + 1)) & mask) ≡ 0) break;
if (j ≡ q[l]) goto up;     /* no support */
t[l++] = 0;     /* add a supported brick */
goto b2;
```

This code is used in section 1.

6. ⟨Advance to the next trial move 6⟩ ≡

```
switch (t[l]) {
case 0: t[l++] = 1;     /* initiate a new layer */
  goto b2;
case 1: if (l + (1 ≪ w[l]) ≤ n) {     /* initiate a new wall */
    k = w[l];
    t[l++] = 2;
    for (j = 0; j < (1 ≪ k); j++) x[l + j] = j + j + 1, p[l + j] = 0, q[l + j] = l, w[l + j] = k + 1, t[l + j] = 0;
    l += j;
    t[l − 1] = 1;
    goto b2;
  }     /* fall through */
case 2: break;
}
up: if (p[l]) {     /* mustn't touch the bottom layer */
  x[l]++;
  if (x[l] ≤ (1 ≪ w[l])) goto b3;
}
```

This code is used in section 1.

7. Index.

⟨ Advance to the next trial move 6 ⟩ Used in section 1.
⟨ Initialize for backtracking 4 ⟩ Used in section 1.
⟨ Process the command line 2 ⟩ Used in section 1.
⟨ Test if a brick at $x[l]$ is supported; if so, add it 5 ⟩ Used in section 1.
⟨ Visit a solution and **goto** *b5* 3 ⟩ Used in section 1.

BACK-KEPLER-TOWERS