1. Intro. Given the values of k, m, n, and a random seed, this little program outputs m uniformly random k-element clauses on n Boolean variables, in the format that my SAT solvers accept. Each clause consists of exactly k literals involving k distinct variables.

More precisely, each of the m clauses is generated by choosing uniformly at random from among the $2^k\binom{n}{k}$ possible clauses. It is possible to generate the same clause more than once, although repetitions are unlikely unless $2^k\binom{n}{k}$ is fairly small or m is fairly large.

(By uniformly random, I mean to within the limits of my 31-bit random number generator.)

The programs SAT-RAND and SAT-RAND-REP-REP, which respectively restrict repetitions more severely and less severely, can be used for comparison.

```
#include <stdio.h>
#include <stdlib.h>
#include "gb_flip.h"
  int k, m, n, seed;    /* command-line parameters */
  main(int argc, char *argv[])
  {
    register int i, j, t, ii, kk, nn;
    ⟨Process the command line 2⟩;
    printf("~␣sat-rand-rep␣%d␣%d␣%d␣%d\n", k, m, n, seed);
    for (j = 0; j < m; j++) ⟨Generate the jth clause 3⟩;
  }
```

2. ⟨Process the command line 2⟩ ≡

```
  if (argc ≠ 5 ∨ sscanf(argv[1], "%d", &k) ≠ 1 ∨ sscanf(argv[2], "%d", &m) ≠ 1 ∨ sscanf(argv[3], "%d",
        &n) ≠ 1 ∨ sscanf(argv[4], "%d", &seed) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣k␣m␣n␣seed\n", argv[0]);
    exit(−1);
  }
  if (k ≤ 0) {
    fprintf(stderr, "k␣must␣be␣positive!\n");
    exit(−2);
  }
  if (m ≤ 0) {
    fprintf(stderr, "m␣must␣be␣positive!\n");
    exit(−3);
  }
  if (n ≤ 0 ∨ n ≥ 100000000) {
    fprintf(stderr, "n␣must␣be␣between␣1␣and␣99999999,␣inclusive!\n");
    exit(−4);
  }
  if (k > n) {
    fprintf(stderr, "k␣mustn't␣exceed␣n!\n");
    exit(−5);
  }
  gb_init_rand(seed);
```

This code is used in section 1.

3. The method of exercise 3.4.2–8(c) is used to generate a random combination of k things from n. (But I changed min to max.)

⟨ Generate the jth clause 3 ⟩ ≡

```
{
  for (kk = k, nn = n; kk; kk--, nn = ii) {
    ⟨ Set ii to the largest in a random kk out of nn 4 ⟩;
    printf("␣%s%d", gb_next_rand() & 1 ? "~" : "", ii);
  }
  printf("\n");
}
```

This code is used in section 1.

4. ⟨ Set ii to the largest in a random kk out of nn 4 ⟩ ≡

```
for (ii = i = 0; i < kk; i++) {
  t = i + gb_unif_rand(nn - i);
  if (t > ii) ii = t;
}
```

This code is used in section 3.

5. Index.

argc: $\underline{1}$, 2.
argv: $\underline{1}$, 2.
exit: 2.
fprintf: 2.
gb_init_rand: 2.
gb_next_rand: 3.
gb_unif_rand: 4.
i: $\underline{1}$.
ii: $\underline{1}$, 3, 4.
j: $\underline{1}$.
k: $\underline{1}$.
kk: $\underline{1}$, 3, 4.
m: $\underline{1}$.
main: $\underline{1}$.
n: $\underline{1}$.
nn: $\underline{1}$, 3, 4.
printf: 1, 3.
seed: $\underline{1}$, 2.
sscanf: 2.
stderr: 2.
t: $\underline{1}$.

$\langle$ Generate the jth clause 3 $\rangle$ Used in section 1.
$\langle$ Process the command line 2 $\rangle$ Used in section 1.
$\langle$ Set ii to the largest in a random kk out of nn 4 $\rangle$ Used in section 3.

SAT-RAND-REP

	Section	Page
Intro	1	1
Index	5	3