1. Intro. This little program finds the parade of rank r from among the $B_{m,n}$ parades that can be made by m girls and n boys, given m, n, and r. (See section 3 of my unpublication "Poly-Bernoulli Bijections.")

```
#define maxn 25   /* Stirling partition numbers will be less than 2^61 */
#include <stdio.h>
#include <stdlib.h>
  int m, n;   /* command-line parameters */
  int gpart, gperm, bpart, bperm;
  long long r, rr;   /* command-line parameter */
  long long spart[maxn + 1][maxn + 1];   /* stirling partition numbers */
  int rgsg[maxn + 1], rgsb[maxn + 1];   /* restricted growth sequences for girls, boys */
  int permg[maxn], permb[maxn];   /* permutations for girls, boys */
  int inv[maxn];   /* inversions of permutation to be constructed */
  main(int argc, char *argv[])
  {
    register int i, j, k, kk;
    register long long f, s, t;
    register double ff, ss, tt;
    ⟨Compute the spart table 2⟩;
    ⟨Process the command line 3⟩;
    ⟨Decompose r 4⟩;
    ⟨Compute and print the result 5⟩;
  }
```

2. ⟨Compute the *spart* table 2⟩ ≡

```
  spart[0][0] = 1;
  for (j = 1; j < maxn; j++)
    for (i = 1; i ≤ j; i++) spart[j][i] = i * spart[j − 1][i] + spart[j − 1][i − 1];
```

This code is used in section 1.

3. ⟨Process the command line 3⟩ ≡

```
  if (argc ≠ 4 ∨ sscanf(argv[1], "%d", &m) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1 ∨ sscanf(argv[3], "%lld",
        &r) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣m␣n␣r\n", argv[0]);
    exit(−1);
  }
  if (m ≥ maxn ∨ n ≥ maxn) {
    fprintf(stderr, "Sorry,␣m␣and␣n␣must␣be␣less␣than␣%d!\n", maxn);
    exit(−2);
  }
```

This code is used in section 1.

4. ⟨ Decompose r 4 ⟩ ≡

```
rr = r;
if (r ≡ 0) kk = 0; else kk = −1, r−−;
for (ss = ff = 1.0, f = 1, k = 1; k ≤ m ∧ k ≤ n; k++) {
  ff ∗= k;    /∗ ff is a floating-point approximation to k! ∗/
  tt = ff ∗ ff ∗ (double) spart[m + 1][k + 1] ∗ (double) spart[n + 1][k + 1];
  ss += tt;
  if (kk < 0) {
    if (tt ≥ (double) #8000000000000000) {
      fprintf(stderr, "I␣don't␣have␣enough␣precision!\n");
      exit(−3);
    }
    f ∗= k;    /∗ f is exactly k! ∗/
    t = f ∗ f ∗ spart[m + 1][k + 1] ∗ spart[n + 1][k + 1];    /∗ t is exactly the kth term ∗/
    if (r < t) kk = k;
    else r −= t;
  }
}
fprintf(stderr, "(B[%d,%d]␣is␣approximately␣%g)\n", m, n, ss);
if (kk < 0) {
  fprintf(stderr, "rank␣%lld␣is␣impossible!\n", rr);
  exit(−4);
}
fprintf(stderr, "We␣will␣find␣the␣parade␣for␣term␣%d␣of␣rank␣%lld.\n", kk, r);
bpart = r % spart[n + 1][kk + 1], r = r/spart[n + 1][kk + 1];
bperm = r % f, r = r/f;
fprintf(stderr, "Boys␣use␣partition␣of␣rank␣%d␣and␣permutation␣of␣rank␣%d.\n", bpart, bperm);
gpart = r % spart[m + 1][kk + 1], gperm = r/spart[m + 1][kk + 1];
fprintf(stderr, "Girls␣use␣partition␣of␣rank␣%d␣and␣permutation␣of␣rank␣%d.\n", gpart, gperm);
```

This code is used in section 1.

5. ⟨ Compute and print the result 5 ⟩ ≡

```
⟨ Compute the partition and permutation for the boys 6 ⟩;
⟨ Compute the partition and permutation for the girls 7 ⟩;
permb[0] = kk + 1;
for (j = 0; j ≤ kk; ) {
  for (i = 1; i ≤ m; i++)
    if (permg[rgsg[i]] ≡ j) printf("␣g%d", i);
  j++;
  for (i = 1; i ≤ n; i++)
    if (permb[rgsb[i]] ≡ j) printf("␣b%d", i);
}
printf("\n");
```

This code is used in section 1.

6. ⟨Compute the partition and permutation for the boys 6⟩ ≡

```
for (i = kk, j = n; j ≥ 0; j−−) {
  if (bpart ≥ (i + 1) ∗ spart[j][i + 1]) bpart −= (i + 1) ∗ spart[j][i + 1], rgsb[j] = i−−;
  else rgsb[j] = bpart/spart[j][i + 1], bpart = bpart % spart[j][i + 1];
}
fprintf(stderr, "Boys␣rgs:");
for (j = 0; j ≤ n; j++) fprintf(stderr, "␣%d", rgsb[j]);
fprintf(stderr, ".\n");
for (j = 1; j ≤ kk; j++) inv[kk + 1 − j] = bperm % j, bperm = bperm/j;
for (j = kk; j; j−−) {
  permb[j] = 1 + inv[j];
  for (i = j + 1; i ≤ kk; i++)
    if (permb[i] ≥ permb[j]) permb[i]++;
}
fprintf(stderr, "Boys␣perm:");
for (j = 1; j ≤ kk; j++) fprintf(stderr, "␣%d", permb[j]);
fprintf(stderr, ".\n");
```

This code is used in section 5.

7. ⟨Compute the partition and permutation for the girls 7⟩ ≡

```
for (i = kk, j = m; j ≥ 0; j−−) {
  if (gpart ≥ (i + 1) ∗ spart[j][i + 1]) gpart −= (i + 1) ∗ spart[j][i + 1], rgsg[j] = i−−;
  else rgsg[j] = gpart/spart[j][i + 1], gpart = gpart % spart[j][i + 1];
}
fprintf(stderr, "Girls␣rgs:");
for (j = 0; j ≤ m; j++) fprintf(stderr, "␣%d", rgsg[j]);
fprintf(stderr, ".\n");
for (j = 1; j ≤ kk; j++) inv[kk + 1 − j] = gperm % j, gperm = gperm/j;
for (j = kk; j; j−−) {
  permg[j] = 1 + inv[j];
  for (i = j + 1; i ≤ kk; i++)
    if (permg[i] ≥ permg[j]) permg[i]++;
}
fprintf(stderr, "Girls␣perm:");
for (j = 1; j ≤ kk; j++) fprintf(stderr, "␣%d", permg[j]);
fprintf(stderr, ".\n");
```

This code is used in section 5.

8. Index.

UNRANK-PARADE1