1. Intro. Given a nonempty parade on the command line, this quick-and-dirty program computes its "rank," as explained in my unpublication *Parades and poly-Bernoulli bijections*.

The rank might be huge. So I don't actually compute it; I produce Mathematica code that will do the numerical work.

(Sorry — I hacked this up in a huge hurry.)

```
#define maxn 100

#include <stdio.h>
#include <stdlib.h>
  int strg[maxn], strb[maxn];     /* the digit strings */
  int d;     /* the order */
  int m, n;     /* how many girls and boys? */
  int perm[maxn], rgs[maxn], hit[maxn];

  main(int argc, char *argv[])
  {
    register int i, j, k, prevj, t, p, l;

    ⟨Process the command line 2⟩;
    ⟨Print the boilerplate to get Mathematica started 3⟩;
    ⟨Figure out the permutation and rgs for the girls 4⟩;
    ⟨Figure out the permutation and rgs for the boys 5⟩;
    printf("extra+((gperm␣brace[%d,%d]+grgs)%d!+bperm)brace[%d,%d]+brgs\n", m + 1, d + 1, d,
        n + 1, d + 1);
  }
```

2. An incorrect command line aborts the run. But we do explain what was wrong.

```
⟨ Process the command line 2 ⟩ ≡
  if (argc < 2) {
    fprintf(stderr, "Usage:␣%s␣<parade>\n", argv[0]);
    exit(−1);
  }
  for (k = 1; k < maxn; k++) strg[k] = strb[k] = −1;
  for (d = 0, k = 1; argv[k]; k++) {
    if (argv[k][0] ≠ 'g' ∧ argv[k][0] ≠ 'b') {
      fprintf(stderr, "Bad␣argument␣'%s';␣should␣start␣with␣g␣or␣b!\n", argv[k]);
      exit(−2);
    }
    for (prevj = j, j = 0, i = 1; argv[k][i] ≥ '0' ∧ argv[k][i] ≤ '9'; i++) j = 10 ∗ j + argv[k][i] − '0';
    if (j ≡ 0 ∨ argv[k][i]) {
      fprintf(stderr, "Bad␣argument␣'%s';␣should␣be␣a␣positive␣number!\n", argv[k]);
      exit(−3);
    }
    if (j ≥ maxn) {
      fprintf(stderr, "Recompile␣me:␣maxn=%d!\n", maxn);
      exit(−6);
    }
    if (argv[k][0] ≡ 'g' ∧ j > m) m = j;
    else if (argv[k][0] ≡ 'b' ∧ j > n) n = j;
    if ((argv[k][0] ≡ 'g' ∧ strg[j] ≥ 0) ∨ (argv[k][0] ≡ 'b' ∧ strb[j] ≥ 0)) {
      fprintf(stderr, "You've␣already␣mentioned␣%s!\n", argv[k]);
      exit(−4);
    }
    if (argv[k][0] ≡ argv[k − 1][0] ∧ prevj > j) {
      fprintf(stderr, "Out␣of␣order:␣%s>%s!\n", argv[k − 1], argv[k]);
      exit(−5);
    }
    if (argv[k][0] ≡ 'b' ∧ argv[k − 1][0] ≠ 'b') d++;
    if (argv[k][0] ≡ 'g') strg[j] = d; else strb[j] = d;
  }
  if (argv[k − 1][0] ≡ 'b') {      /∗ parade ended with a boy: d is too large ∗/
    for (j = 1; j ≤ n; j++)
      if (strb[j] ≡ d) strb[j] = 0;
    d−−;
  }
  for (j = 1; j ≤ m; j++)
    if (strg[j] < 0) {
      fprintf(stderr, "girl␣g%d␣is␣missing!\n", j);
      exit(−7);
    }
  for (j = 1; j ≤ n; j++)
    if (strb[j] < 0) {
      fprintf(stderr, "boy␣b%d␣is␣missing!\n", j);
      exit(−8);
    }
  fprintf(stderr, "OK,␣that's␣a␣valid␣parade␣of␣order␣%d␣with␣%d␣girls␣and␣%d␣boys!\n", d, m, n);
```

This code is used in section 1.

3. ⟨Print the boilerplate to get Mathematica started 3⟩ ≡

```
printf("(*␣output␣from␣%s", argv[0]);
for (k = 1; argv[k]; k++) printf("␣%s", argv[k]);
printf("␣*)\n");
printf("brace=StirlingS2\n");
printf("prank[inv_]:=Block[{sum=0,n=Length[inv]},\n");
printf("␣For[j=1,j<n,j++,sum=(sum+inv[[j]])*(n-j)];sum]\n");
printf("srank[rgs_]:=Block[{sum=0,max=0,n=Length[rgs]},\n");
printf("␣For[j=1,j<=n,j++,\n");
printf("␣␣If[rgs[[j]]>max,max++;sum+=(max+1)brace[j,max+1],\n");
printf("␣␣␣sum+=rgs[[j]]brace[j,max+1]]];\n");
printf("␣sum]\n");
printf("extra␣=␣Sum[k!^2*brace[%d+1,k+1]*brace[%d+1,k+1],{k,0,%d}]\n", m, n, d − 1);
```

This code is used in section 1.

4. ⟨Figure out the permutation and rgs for the girls 4⟩ ≡

```
for (j = 1; j ≤ d; j++) hit[j] = −1;
for (k = 0, j = 1; j ≤ m; j++) {
  if (hit[strg[j]] < 0) hit[strg[j]] = ++k, perm[k] = strg[j];
  rgs[j] = hit[strg[j]];
}
fprintf(stderr, "girls'␣rgs␣is");
for (j = 0; j ≤ m; j++) fprintf(stderr, "␣%d", rgs[j]);
fprintf(stderr, "\nand␣their␣permutation␣is");
for (j = 1; j ≤ d; j++) fprintf(stderr, "␣%d", perm[j]);
fprintf(stderr, "\n");
printf("gperm=prank[{");
for (j = 1; j ≤ d; j++) {
  if (j > 1) printf(",");
  for (k = 0, i = j + 1; i ≤ d; i++)
    if (perm[i] < perm[j]) k++;
  printf("%d", k);
}
printf("}]\ngrgs=srank[{");
for (j = 1; j ≤ m; j++) {
  if (j > 1) printf(",");
  printf("%d", rgs[j]);
}
printf("}]\n");
```

This code is used in section 1.

5. ⟨Figure out the permutation and rgs for the boys 5⟩ ≡

```
for (j = 1; j ≤ d; j++) hit[j] = −1;
for (k = 0, j = 1; j ≤ n; j++) {
  if (hit[strb[j]] < 0) hit[strb[j]] = ++k, perm[k] = strb[j];
  rgs[j] = hit[strb[j]];
}
fprintf(stderr, "boys'␣rgs␣is");
for (j = 0; j ≤ n; j++) fprintf(stderr, "␣%d", rgs[j]);
fprintf(stderr, "\nand␣their␣permutation␣is");
for (j = 1; j ≤ d; j++) fprintf(stderr, "␣%d", perm[j]);
fprintf(stderr, "\n");
printf("bperm=prank[{");
for (j = 1; j ≤ d; j++) {
  if (j > 1) printf(",");
  for (k = 0, i = j + 1; i ≤ d; i++)
    if (perm[i] < perm[j]) k++;
  printf("%d", k);
}
printf("}]\nbrgs=srank[{");
for (j = 1; j ≤ n; j++) {
  if (j > 1) printf(",");
  printf("%d", rgs[j]);
}
printf("}]\n");
```

This code is used in section 1.

6. Index.

RANK-PARADE1