1. Intro. I'm experimenting with a novel way to represent permutations, and applying it to Langford's problem. The latter problem can be regarded as the task of creating a permutation p of $\{1,2,\ldots,2n\}$ with the property that $p_i = k$ implies $p_{i+n} = k+i+1$, for $1 \le i \le n$. (It means that we put the digit i into positions k and $k+i+1$. This model for the problem was studied by Gent, Miguel, and Rendl in *LNCS* **4612** (2007), 184–199.)

The permutation representation uses *order encoding* in two dimensions: We have variables y_{ij} meaning that $p_i \le j$ and z_{ij} meaning that $q_j \le i$, where q is the inverse of p. The permutation p is implicit; we have $p_i = k$ if and only if $y_{ik} = 1$ and $y_{i(k-1)} = 0$ if and only if $z_{ik} = 1$ and $z_{(i-1)k} = 0$. The boundary conditions are $y_{i0} = z_{0j} = 0$ and $y_{in} = z_{nj} = 1$. Also $y_{i(j-1)} \le y_{ij}$ and $z_{(i-1)j} \le z_{ij}$.

```
#include <stdio.h>
#include <stdlib.h>
  int n;  /* command-line parameter */
  main(int argc, char *argv[])
  {
    register int i, j, k, nn;
    ⟨ Process the command line 2 ⟩;
    ⟨ Generate the monotonicity clauses 3 ⟩;
    ⟨ Generate the clauses that relate y's to z's 4 ⟩;
    ⟨ Generate the clauses for Langford's problem 5 ⟩;
  }
```

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);
  }
  nn = n + n;
```

This code is used in section 1.

3. ⟨ Generate the monotonicity clauses 3 ⟩ ≡

```
  for (i = 1; i ≤ nn; i++) {
    printf("~%dy%d\n", i, 0);
    printf("~%dz%d\n", 0, i);
    printf("%dy%d\n", i, nn);
    printf("%dz%d\n", nn, i);
  }
  for (i = 1; i ≤ nn; i++)
    for (j = 1; j ≤ nn; j++) {
      printf("~%dy%d␣%dy%d\n", i, j − 1, i, j);
      printf("~%dz%d␣%dz%d\n", i − 1, j, i, j);
    }
```

This code is used in section 1.

4. We can derive the following clauses by imagining a matrix with $x_{ij} = [p_i = j]$ and eliminating the x variables.

```
⟨ Generate the clauses that relate y's to z's 4 ⟩ ≡
  for (i = 1; i ≤ nn; i++)
    for (j = 1; j ≤ nn; j++) {
      printf("~%dy%d␣%dz%d␣~%dz%d\n", i, j − 1, i − 1, j, i, j);
      printf("%dy%d␣%dz%d␣~%dz%d\n", i, j, i − 1, j, i, j);
      printf("%dy%d␣~%dy%d␣~%dz%d\n", i, j − 1, i, j, i − 1, j);
      printf("%dy%d␣~%dy%d␣%dz%d\n", i, j − 1, i, j, i, j);
    }
```

This code is used in section 1.

5.

```
⟨ Generate the clauses for Langford's problem 5 ⟩ ≡
  for (i = 1; i ≤ n; i++) {
    printf("%dy%d\n", i, nn − 1 − i);
    printf("~%dy%d\n", i + n, i + 1);
  }
  for (i = 1; i ≤ n; i++) {
    for (j = 1; j ≤ nn − 1 − i; j++) {
      printf("%dy%d␣~%dy%d␣~%dy%d\n", i, j − 1, i, j, i + n, i + j);
      printf("%dy%d␣~%dy%d␣%dy%d\n", i, j − 1, i, j, i + n, i + j + 1);
    }
    for (j = i + 2; j ≤ nn; j++) {
      printf("%dy%d␣~%dy%d␣~%dy%d\n", i + n, j − 1, i + n, j, i, j − i − 2);
      printf("%dy%d␣~%dy%d␣%dy%d\n", i + n, j − 1, i + n, j, i, j − i − 1);
    }
  }
```

This code is used in section 1.

6. Index.

SAT-NEWLANGFORD