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 $1 \leq i \leq n$. (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 $p_i = j$ and $z_{ij}$ meaning that $q_j = i$, 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 $z_{ik} = 0$ if and only if $y_{i(k-1)} = 1$ and $z_{i-1}k = 0$. The boundary conditions are $y_{i0} = z_0 = 0$ and $y_{in} = z_n = 1$. Also $y_{i(j-1)} \leq y_{ij}$ and $z_{i-1}j \leq z_{ij}.$

```c
#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
", 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
", i, 0);
    printf("%dy%d\n", 0, i);
    printf("%dy%d\n", i, nn);
    printf("%dy%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("%dy%d, %dy%d\n", i, j - 1, j, i);
    }
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.

\[
\langle \text{Generate the clauses that relate } y \text{'s to } z \text{'s} \rangle
\]

\[
\text{for } (i = 1; i \leq mn; i++)
\]

\[
\text{for } (j = 1; j \leq mn; j++)
\]

\[
\text{printf } ("\%dy\%d\%dz\%d\%dz\%d\n", i, j-1, i-1, j, i, j);
\]

\[
\text{printf } ("\%dy\%d\%dz\%d\%dz\%d\n", i, j-1, i, j, i, j);
\]

\[
\text{printf } ("\%dy\%d\%dz\%d\%dz\%d\n", i, j-1, i, j, i, j);
\]

\[
\text{printf } ("\%dy\%d\%dz\%d\%dz\%d\n", i, j-1, i, j, i, j);
\]

This code is used in section 1.

5. \( \langle \text{Generate the clauses for Langford’s problem} \rangle \equiv \)

\[
\text{for } (i = 1; i \leq n; i++)
\]

\[
\text{printf } ("\%dy\%d\n", i, nn - 1 - i);
\]

\[
\text{printf } ("\%dy\%d\n", i + n, i + 1);
\]

\[
\text{for } (j = 1; j \leq nn - 1 - i; j++)
\]

\[
\text{printf } ("\%dy\%d\%dy\%d\%dy\%d\n", i, j-1, i, j, i + n, i + j);
\]

\[
\text{printf } ("\%dy\%d\%dy\%d\%dy\%d\n", i, j-1, i, j, i + n, i + j + 1);
\]

\[
\text{for } (j = i + 2; j \leq nn; j++)
\]

\[
\text{printf } ("\%dy\%d\%dy\%d\%dy\%d\%dy\%d\n", i + n, j-1, i + n, j, i + n, j - i - 2);
\]

\[
\text{printf } ("\%dy\%d\%dy\%d\%dy\%d\%dy\%d\n", i + n, j-1, i + n, j, i + n, j - i - 1);
\]

This code is used in section 1.
6. Index.

argc: 1, 2.
argv: 1, 2.
ext: 2.
fprintf: 2.
i: 1.
j: 1.
k: 1.
main: 1.
n: 1.
nn: 1, 2, 3, 4, 5.
printf: 3, 4, 5.
sscanf: 2.
stderr: 2.
(Generate the clauses for Langford’s problem 5) Used in section 1.
(Generate the clauses that relate y’s to z’s 4) Used in section 1.
(Generate the monotonicity clauses 3) Used in section 1.
(Process the command line 2) Used in section 1.
SAT-NEWLANGFORD

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro</td>
<td>1</td>
</tr>
<tr>
<td>Index</td>
<td>6</td>
</tr>
</tbody>
</table>

1 1
3 3