(See https://cs.stanford.edu/~knuth/programs.html for date.)

1. Intro. This program computes the $2 \times n$ whirlpool permutation that corresponds to a given up-up-or-down-down permutation of $\{1, 2, \ldots, 2n-1\}$, which appears on the command line. So it's essentially the inverse of the program WHIRLPOOL2N-ENCODE, and its output is a suitable input to that program.

By "up-up-or-down-down permutation" of length $2n-1$, I mean a permutation $p_1 \ldots p_{2n-1}$ such that $p_{2k-1} < p_{2k}$ if and only if $p_{2k} < p_{2k+1}$, for $1 \le k < n$.

(I've made no attempt to be efficient.)

(But I didn't go out of my way to be inefficient.)

(Apologies for doing this hurriedly.)

```
#define maxn 100
#include <stdio.h>
#include <stdlib.h>
  int a[2 * maxn];    /* where we build the answer */
  int g[2 * maxn];    /* the given permutation */
  int w[2 * maxn];    /* workspace */
  int used[2 * maxn];
  main(int argc, char *argv[])
  {
    register int i, j, k, n, nn, t, x, y, saven;
    ⟨Process the command line 2⟩;
    ⟨Prepare to grow 4⟩;
    for (n = 1; n < saven; n++) ⟨Grow the solution from n to n + 1 5⟩;
    ⟨Print the answer 6⟩;
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
if (argc & 1) {
  fprintf(stderr, "Usage:␣%s␣a1␣a2␣...␣a(2n-1)\n", argv[0]);
  exit(−1);
}
nn = argc, n = saven = nn/2;
if (n > maxn) {
  fprintf(stderr, "Recompile␣me:␣This␣program␣has␣maxn=%d!\n", maxn);
  exit(−99);
}
for (k = 1; k < nn; k++) {
  if (sscanf(argv[k], "%d", &g[k]) ≠ 1) {
    fprintf(stderr, "Bad␣perm␣element␣`%s'!\n", argv[k]);
    exit(−2);
  }
  if (g[k] ≤ 0 ∨ g[k] ≥ nn) {
    fprintf(stderr, "Perm␣element␣`%d'␣out␣of␣range!\n", g[k]);
    exit(−3);
  }
  if (used[g[k]]) {
    fprintf(stderr, "Duplicate␣perm␣entry␣`%d'!\n", g[k]);
    exit(−4);
  }
  used[g[k]] = 1;
}
⟨ Verify the up-up-or-down-down criteria 3 ⟩;
```

This code is used in section 1.

3. ⟨ Verify the up-up-or-down-down criteria 3 ⟩ ≡

```
for (k = 2; k < nn; k += 2) {
  if ((g[k − 1] < g[k]) ≠ (g[k] < g[k + 1])) {
    fprintf(stderr, "Not␣up-up-or-down-down!␣(%d␣%d␣%d)\n", g[k − 1], g[k], g[k + 1]);
    exit(−6);
  }
}
```

This code is used in section 2.

4. Here I compress the "uncompressed" numbers in the given permutation.

```
⟨Prepare to grow 4⟩ ≡
  $a[0] = 1$;
  for ($k = 1$; $k < nn$; $k$++) $used[k] = 0$;
  for ($k = 2$; $k < nn$; $k$ += 2) {
    $x = g[k-1], y = g[k]$;
    for ($t = 0, j = 1$; $j < x$; $j$++)
      if ($used[j]$) $t$++;
    $g[k-1]$ −= $t$;
    for ($t = 0, j = 1$; $j < y$; $j$++)
      if ($used[j]$) $t$++;
    $g[k]$ −= $t$;
    $used[x] = used[y] = 1$;
  }
  $g[nn-1] = 1$;
```

This code is used in section 1.

5.

```
⟨Grow the solution from $n$ to $n+1$ 5⟩ ≡
  {
    $x = g[nn-n-n-1], y = g[nn-n-n]$;
    $t = y - (x < y \mathbin{?} 2 : 1) - a[0] + n + n$;
    for ($k = n-1$; $k \ge 0$; $k$−−)
      $a[k+1] = (a[k]+t) \,\%\, (n+n), a[k+saven+1] = (a[k+saven]+t) \,\%\, (n+n)$;
    for ($k = 1$; $k \le n$; $k$++) $a[k]$ += $(a[k] < x-1 \mathbin{?} 1 : 2)$, $a[k+saven]$ += $(a[k+saven] < x-1 \mathbin{?} 1 : 2)$;
    $a[0] = x$;
  }
```

This code is used in section 1.

6.

```
⟨Print the answer 6⟩ ≡
  for ($k = 0$; $k < nn$; $k$++) printf("␣%d", $a[k]$);
  printf("\n");
```

This code is used in section 1.

7. Index.

WHIRLPOOL2N-DECODE