1. Intro. This program solves exercise 6.2.2–50 of *The Art of Computer Programming* (which was added to Volume 3 in September, 2021, just in time for the 43rd printing!). Here's the statement of that exercise:

> **50.** [30] Let $p_1p_2\ldots p_n$ be a permutation of $\{1,2,\ldots,n\}$. Suppose the values p_1, p_2, …, p_n have been inserted successively into an initially empty binary tree using Algorithm T, but with $\mathtt{Q}\gets K$ in step T5 when storing key K. Explain how to compute all of the resulting links `LLINK(`k`)` and `RLINK(`k`)` for $1\le k\le n$ in just $O(n)$ steps. (For example, the permutation 3142 would yield $(\mathtt{LLINK(1)},\ldots,\mathtt{LLINK(4)})=(\Lambda,\Lambda,1,\Lambda)$ and $(\mathtt{RLINK(1)},\ldots,\mathtt{RLINK(4)})=(2,\Lambda,4,\Lambda)$.)

The following solution, suggested by Robert E. Tarjan, implicitly uses the one-to-one correspondence between binary search trees and binary tournaments in §3 of Jean Vuillemin's classic paper "Cartesian trees," *Communications of the ACM* **23** (1980), 229–239. (Stating this another way, it implicitly uses the fact that the binary search tree defined by a permutation has the same shape as the "increasing binary tree" defined by the *inverse* of that permutation. The increasing binary tree defined by a permutation retains symmetric order, but forces all paths from the root to be increasing.)

2. The given permutation should appear on *stdin*, as the sequence of numbers $p_1\ p_2\ \ldots\ p_n$, separated by whitespace. The output on *stdout* will show the root, followed on separate lines by the links of 1, 2, …, n.

```
#define maxn 1024
#define panic(m,k)
        { fprintf(stderr,"%s!␣(%d)\n",m,k); exit(−666); }
#define pan(m)
        { fprintf(stderr,"%s!\n",m); exit(−66); }
#include <stdio.h>
#include <stdlib.h>
  int p[maxn + 2];   /* the given permutation */
  int q[maxn + 2];   /* its inverse */
  int stack[maxn + 1];   /* the working stack */
  int stackx[maxn + 1];   /* indexes associated with the working stack */
  int llink[maxn + 2], rlink[maxn + 1];   /* the answers */
  int inx;   /* a place for input data from fscanf */
  void main(void)
  {
    register int i, j, k, m, n, s;
    ⟨Input the permutation 3⟩;
    ⟨Compute the links 4⟩;
    ⟨Output the links 6⟩;
  }
```

3. Input. Let's get the boring stuff out of the way. Our first task is to input the permutation, and check that it makes sense.

⟨Input the permutation 3⟩ ≡
for $(m = n = 0;\ fscanf(stdin, \texttt{"\%d"}, \&inx) \equiv 1;\ n{+}{+})$ {
  if $(inx \le 0 \lor inx > maxn)$ *panic*("element␣out␣of␣range", *inx*);
  if $(inx > m)$ $m = inx$;
  $p[n+1] = inx$;
}
if $(n \equiv 0)$ *pan*("the␣permutation␣must␣have␣at␣least␣one␣element");
if $(m > n)$ *panic*("too␣few␣elements", $m - n$);
if $(m > n)$ *panic*("too␣many␣elements", $n - m$);
for $(k = 1;\ k \le n;\ k{+}{+})$ $q[p[k]] = k$;   /* compute the inverse */
for $(k = 1;\ k \le n;\ k{+}{+})$
  if $(q[k] \equiv 0)$ *panic*("missing␣element", k);

This code is used in section 2.

4. Doin' it. During this algorithm, which is amazingly short and sweet, we'll have $0 = stack[0] < stack[1] < \cdots < stack[s]$, where the stack elements will be a subsequence of $q[1]$, $q[2]$, ..., $q[n]$. If $stack[t]$ came from $q[k]$, $stackx[t]$ will be k.

The basic idea is that every element will be pointed to by one link. As soon as we know that some node i will point to another node j, we store that link and essentially remove j from the system. (In other words, we compute the tree bottom-up.)

We assume that the *llink* and *rlink* arrays are initially zero, and that zero represents a null link.

$\langle$ Compute the links 4 $\rangle \equiv$

$stack[0] = stackx[0] = 0;$
$stack[1] = q[1], stackx[1] = 1, s = 1;$
$q[n+1] = 0;$
for $(k = 2;\ s > 0 \lor k \le n;\ )$ {
  if $(stack[s] < q[k])$ $s{+}{+}, stack[s] = q[k], stackx[s] = k{+}{+};$
  else if $(stack[s-1] > q[k])$ $s{-}{-}, rlink[stackx[s]] = stackx[s+1];$
  else $s{-}{-}, llink[k] = stackx[s+1];$
}

This code is used in section 2.

5. Curiously, the very same algorithm (in slight disguise) was published on page 317 of a paper by Johnson M. Hart ["Fast recognition of Baxter permutations using syntactical and complete bipartite composite dag's," *International Journal of Computer and Information Sciences* **9** (1980), 307–321] — but not in the context of binary trees. His context was "complete bipartite composite digraphs," which are a convoluted way of formalizing floorplans(!).

6. Output. Finally s will become zero when $k = n + 1$, because $q[n + 1] = 0$.

$\langle$ Output the links 6 $\rangle \equiv$

```
printf("The root is %d.\n", llink[n + 1]);
for (k = 1; k ≤ n; k++) {
  printf("%5d:", k);
  if (llink[k]) printf("%5d,", llink[k]);
  else printf("   /\\,");
  if (rlink[k]) printf("%5d.\n", rlink[k]);
  else printf("   /\\.\n");
}
```

This code is used in section 2.

7. Index.

OFFLINE-TREE-INSERTION