**1. Introduction.** This short program implements a Françon-inspired bijection between binary trees with Strahler number $s$ and nested strings with height $h$, where $2^s-1 \le h < 2^{s+1}-1$. But it uses a direct method that is complementary to his approach. [Reference: Jean Françon, "Sur le nombre de registres nécessaires a l'évaluation d'une expression arithmétique," *R.A.I.R.O. Informatique théorique* **18** (1984), 355–364.]

```
#define n 17   /* nodes in the tree */
#define nn (n + n)
#include <stdio.h>
  int d[nn + 1];   /* the path, a sequence of ±1s */
  int l[n + 1], r[n + 1];   /* tree links */
  int h[nn + 1], q[n + 1], qm[n + 1];   /* heap and queue structures for decision-making */
  int serial;   /* total number of cases checked */
  int count[10];   /* individual counts by Strahler number */
  ⟨Subroutines 5⟩
  main()
  {
    register int i, j, k, jj, kk, m, p, s;
    printf("Checking␣binary␣trees␣with␣%d␣nodes...\n", n);
    ⟨Set up the first nested string, d 2⟩;
    while (1) {
      ⟨Find the tree corresponding to d 7⟩;
      ⟨Check the Strahler number 4⟩;
      ⟨Check the inverse bijection 9⟩;
      ⟨Move to the next nested string, or goto done 3⟩;
    }
  done:
    for (s = 1; count[s]; s++)
      printf("Altogether␣%d␣cases␣with␣Strahler␣number␣%d.\n", count[s], s);
  }
```

**2.** Nested strings (aka Dyck words) are conveniently generated by Algorithm 7.2.1.6P of *The Art of Computer Programming*.

```
⟨Set up the first nested string, d 2⟩ ≡
  for (k = 0; k < nn; k += 2) d[k] = +1, d[k + 1] = −1;
  d[nn] = −1, i = nn − 2;
```

This code is used in section 1.

**3.** At this point, variable $i$ is the position of the rightmost '+1' in $d$.

```
⟨Move to the next nested string, or goto done 3⟩ ≡
  d[i] = −1;
  if (d[i − 1] < 0) d[i − 1] = 1, i−−;
  else {
    for (j = i − 1, k = nn − 2; d[j] > 0; j−−, k −= 2) {
      d[j] = −1, d[k] = +1;
      if (j ≡ 0) goto done;
    }
    d[j] = +1, i = nn − 2;
  }
```

This code is used in section 1.

**4.** ⟨ Check the Strahler number 4 ⟩ ≡

```
for (s = j = k = 1; k < nn - 1; j += d[k], k++)
  if (j >= ((1 << s) - 1)) s++;
s--;   /* now s is the Strahler number */
count[s]++, serial++;
if (strahler(1) != s) {
  fprintf(stderr, "I goofed on case %d.\n", serial);
}
```

This code is used in section 1.

**5.** ⟨ Subroutines 5 ⟩ ≡

```
int strahler(int x)
{
  register int sl, sr;
  if (l[x]) sl = strahler(l[x]);
  else sl = 0;
  if (r[x]) sr = strahler(r[x]);
  else sr = 0;
  return (sl > sr ? sl : sl < sr ? sr : sl + 1);
}
```

This code is used in section 1.

**6. The main algorithm.** A large family of bijections between nested strings and binary trees was described by Proskurowski in *JACM* **27** (1980), page 1: We build a binary tree by choosing, at each step, some node and some yet-unset link field in that node; then we look at the next element $d[p]$ of the nested string. The link is set to a new node if $d[p] > 0$, and to null if $d[p] < 0$. The bijection implemented here is of that type.

To decide what link should be constructed next, we use a heap-like data structure $h[1]$, $h[2]$, …, in which cell $k$ is the parent of cells $2k$ and $2k+1$. The cell elements are pointers to nodes in the tree being built, and the nodes recorded in the heap can be embedded as a subtree of that tree. (In other words, if $h[k]$ and $h[\lfloor k/2\rfloor]$ are both nonzero, they point to nodes of the tree in which the first is a descendant of the second. It might be helpful to imagine a set of pebbles on the tree, with the heap cells recording the positions of those pebbles.) When $h[2k]=0$, meaning that heap cell $2k$ is empty, we also have $h[2k+1]=0$. The basic idea of the algorithm is to attempt to fill the first empty cell $k$ in the heap, by setting the links of the tree node pointed to by $h[k/2]$.

The number of elements in the heap is always the partial sum $d[0]+\cdots+d[p]$. If this number is $2^t-1$ or more, the Strahler number of the binary tree is at least $t$. Conversely, if the Strahler number is $s$, one can show without difficulty that the partial sum will indeed reach the value $2^s-1$ at some point, with the heap at that time containing the “topmost” complete subtree of size $2^s-1$ embedded in the tree.

For validity of this algorithm, we don't really need to choose the first hole in the heap. Any rule for choosing $k$ would work, provided only that (a) $k$ is even; (b) $h[k/2]\neq 0$; and (c) $k \geq 2^t$ implies $d[0]+\cdots+d[p]\geq 2^t-1$. Thus there are many possible bijections, some of which are presumably easier to analyze than others.

**7.** Variable $m$ represents the number of nodes in the tree; variable $p$ is our position in the nested string; and variable $k$ is a lower bound on the location of the least hole in the heap.

⟨ Find the tree corresponding to $d$ 7 ⟩ ≡

```
h[1] = m = 1, k = 2, p = 0;
while (1) {
  while (h[k]) k += 2;      /* find the smallest hole */
  kk = h[k >> 1];      /* kk is the node pointed to by k's parent */
  if (d[++p] > 0) h[k] = l[kk] = ++m; else l[kk] = 0;
  if (d[++p] > 0) h[k + 1] = r[kk] = ++m; else r[kk] = 0;
  if (h[k]) {
    if (h[k + 1]) continue;
    kk = k;
  } else if (h[k + 1]) kk = k + 1;
  else {
    h[k >> 1] = 0, kk = (k >> 1) ⊕ 1, k = kk & −2;
    if (k ≡ 0) break;      /* we're done when the heap is empty */
  }
  ⟨ Move the subheap rooted at kk up one level 8 ⟩;
}
```

This code is used in section 1.

**8.** Let the binary representation of $kk$ be $(b_t \ldots b_0)_2$. We want to set $h[(b_t \ldots b_1\alpha)_2] \leftarrow h[(b_t \ldots b_0\alpha)_2]$ for all binary strings $\alpha$.

$\langle$ Move the subheap rooted at $kk$ up one level 8 $\rangle \equiv$

$j = 0, jj = 1, q[0] = kk, qm[0] = 1;$
**while** $(j < jj)$ {
  $kk = q[j];$
  $h[((kk \gg 1) \,\&\, -qm[j]) + (kk \,\&\, (qm[j] - 1))] = h[kk];$
  **if** $(h[kk + kk])$ $q[jj] = kk + kk, q[jj + 1] = kk + kk + 1, qm[jj] = qm[jj + 1] = qm[j] \ll 1, jj \mathrel{+}= 2;$
  **else** $h[kk] = 0;$
  $j{+}{+};$
}

This code is used in sections 7 and 9.

**9. The inverse algorithm.** To reverse the process, we simply look at the tree and build the nested string, instead of vice versa. The same heap-oriented logic applies.

```
#define check(s)
        { if (d[++p] ≠ s) fprintf(stderr, "Rejection␣at␣position␣%d␣of␣case␣%d!\n", p, serial); }
⟨Check the inverse bijection 9⟩ ≡
  h[1] = 1, k = 2, p = 0;
  while (1) {
    while (h[k]) k += 2;     /* find the smallest hole */
    kk = h[k ≫ 1];     /* kk is the node pointed to by k's parent */
    if (l[kk]) {
      h[k] = l[kk]; check(+1);
    } else check(−1);
    if (r[kk]) {
      h[k + 1] = r[kk]; check(+1);
    } else check(−1);
    if (h[k]) {
      if (h[k + 1]) continue;
      kk = k;
    } else if (h[k + 1]) kk = k + 1;
    else {
      h[k ≫ 1] = 0, kk = (k ≫ 1) ⊕ 1, k = kk & −2;
      if (k ≡ 0) break;     /* we're done when the heap is empty */
    }
    ⟨Move the subheap rooted at kk up one level 8⟩;
  }
```

This code is used in section 1.

**10. Index.**

# FRANÇON