Copyright Notice

The manuscript was published as

Acta Informatica 6 (1976): 357–359.

On a gauntlet thrown by David Gries.

by

Edsger W.Dijkstra

It is requested to design a program that will generate the N! permutations of the values from 0 through N-1 in such an order that the transition from one permutation to the next is always performed by exactly one swap of two neighbours.

In a permutation each pair of values such that the larger value precedes the smaller one, presents a so-called "inversion". (In particular: the one and only permutation with zero inversions is the one in which the values are placed in monotonically increasing order.) The notion of inversions can be expected to be relevant because the swapping of two neighbours changes the total number of inversions by (plus or minus) 1 , and it is, therefore, suggested to characterize each permutation by its inversions. This can be done by introducing N inversion counters inv[i] for $0 \leq i < N$, where inv[i] equals the number of inversions between the value i and the values smaller than i . (From this definition $0 \leq inv[i] \leq i$ follows; the total number of inversions of a permutation is the sum of the corresponding inv[i]-values.) That each permutation defines the inv[i]-values uniquely is obvious; that the inv[i]-values define the permutation uniquely is easily seen by considering the algorithm constructing the permutation from the inv[i]-values --processing these values in the order of increasing i -- : this algorithm leaves us no choice.

There is, therefore, a one-to-one correspondence between the N! possible inv-values and the N! permutations, and the question becomes, which modification of the inv-value correspond to a swap of neighbours: each swap of two neighbours changes exactly one inv[i]-value by 1 , viz. with i = the larger of the two values swapped. The value of inv[i] is to be increased by 1 if the swap increases the number of inversions --i.e. if the larger value moves "to the left"-- otherwise it is to be decreased.

A feasible sequence of inv-values to be generated is now reasonably obvious; it is the generalization of the Gray-code. For N = 4 it begins:

inv[0]	inv[1]	inv[2]	inv[3]	a[0]	a[1]	a[2]	a[3]
0	0	0	0	0	1	2	3
0	0	0	1	0	1	3	2
0	0	0	2	0	3	1	2
0	0	0	3	3	0	1	2
0	0	1	3	3	0	2	1
0	0	1	2	0	3	2	1
0	0	1	1	0	2	3	1
0	0	1	0	0	2	1	3
0	0	2	0	2	0	1	3
0	0	2	1	2	0	3	1
0	0	2	2	2	3	0	1
0	0	2	3	3	2	0	1
0	1	2	3	3	2	1	0
0	1	2	2	2	3	1	0

etc.

The rule is as follows. An inv[i]-value is defined to be "changeable" if it may be increased or decreased by 1 . It may be increased if the sum of the inv[j]-values to its left --i.e. for $j < i$ -- is even and it has not reached its maximum value; it may be decreased if the sum of the inv[j]-values to its left is odd and it has not reached its minimum value zero. At each step, always the right-most changeable inv[i]-value --i.e. with the largest i-value-- is changed. It is not difficult to see that in the permutation, the value i is, indeed swapped with a smaller value. (We leave it to the reader to prove that, starting with $inv[i] = 0$ for all i , N!-1 applications of this rule yields the remaining N!-1 possible inv-values.)

After having established the value i , such that inv[i] has to be changed, and, also, whether the value i has to be swapped with its predecessor in the permutation (corresponding to an increase of inv[i]) or with its successor in the permutation (corresponding to a decrease of inv[i]), we have to establish the place c in the permutation, where the value i is located. This value c is given by

i - (the number of smaller values to its right)
  + (the number of larger values to its left) .

Because for all $j > i$, inv[j] has an extreme value (0 or j), c is given by

c = i - inv[i] + (the number of values of j such that j > i and inv[j] = j)

In the following program we have given inv[0] --which should be constantly = 0 -- the funny value -2 ; this is the usual, mean, little coding trick, in order to let the search for the right-most changeable inv[i]-value terminate artificially when there is no more such an inv[i]-value. The variable "totinv" records the total number of inversions in the array a , used to record the permutation. The invariant of the inner loop in the following program is

$0 \leq i < N$ and
$(i < j < N) \Rightarrow$ "inv[j] is not changeable" and
linv = "sum of the inv[j]-values for $0 < j < i$" and
c = "number of values j such that $i < j < N$ and inv[j] = j" .

```
begin integer array a, inv [0:N-1]; boolean ready; integer totinv, i, c, linv;
i:= 0; do i < N → a[i]:= i; inv[i]:= 0; i:= i + 1 od; inv[0]:= -2;
ready:= false; totinv:= 0;
do non ready → printarray(a);
      {establish the value i such that inv[i] has to be changed:}
      i:= N - 1; c:= 0; linv:= totinv - inv[i];
      do inv[i] = i and even(linv) →
           c:= c + 1; i:= i - 1; linv:= linv - inv[i]
       [] inv[i] = 0 and odd(linv) →
           i:= i - 1; linv:= linv - inv[i]
      od { inv[i] has to be changed if i > 0 };
      {compute the position c of value i in the permutation:}
      c:= c + i - inv[i];
      {generate next permutation or signal completion:}
      if even(linv) and i > 0 →
           inv[i]:= inv[i] + 1; totinv:= totinv + 1; swap(a, c-1, c)
       [] odd(linv) and i > 0 →
           inv[i]:= inv[i] - 1; totinv:= totinv - 1; swap(a, c, c+1)
       [] i = 0 → ready:= true
      fi
od
end
```

Post Scriptum.

The problem solved above has been posed to me by David Gries, who told me that he had found it a nontrivial task to present a solution to it in a convincing manner. The argument shown led quickly to the above solution, which is submitted for publication upon his request. Finally I would like to express my great indebtness to the referee whose many and well-considered recommendations have led to a considerable improvement in readability. (End of Post Scriptum.)

Burroughs
Plataanstraat 5
NUENEN - 4565
The Netherlands

prof.dr.Edsger W.Dijkstra