On a gauntlet thrown by David Gries.

Freshly arrived from the U.S.A. --they still struggled with the time shift-- David Gries and his family paid us a visit a few weeks ago. It was not an occasion for "working together", yet some shoptalk was unavoidable, and David confronted me with the problem how to generate the n! permutations of the numbers 0 through n-1 such that the transition from one permutation to the next would only involve the swapping of two neighbours. He told me the problem because he had found it a non-trivial task to present an algorithm solving this problem in a convincing manner. He --good boy!-- respected my desire not to be told how he had solved the difficulties of presenting his solution, and he granted me the opportunity to think about the problem myself. (He told me, that the problem had been dealt with in the Algorithms Section of the Comm.ACM, but at the level of unintelligibility that is characteristic for that Section, and we agreed wholeheartedly that a tradition of clearer presentation of algorithms is most sorely needed.)

Note. Those of my readers who would like to try to solve this problem themselves should stop reading here. (End of Note.)

As the swapping of two neighbours changes the number of inversions --i.e. the number of pairs in the wrong order-- by 1 , it is suggested to try to characterize each permutation by its inversions. If we consider an arbitrary permutation of the numbers 0 through n-1 , each permutation is uniquely characterized by the values inv(i) , with $0 \leq \mathrm{inv}(i) \leq i$ for $0 \leq i < n$, where inv(i) equals the number of numbers < i , that are placed at "the wrong side" of i ; inv(i) = the number of inversions between the value i and smaller values. (The total number of inversions of the permutation is the sum of all the inv(i)-values.) That each permutation defines the inv(i)-values uniquely is obvious; that the inv(i)-values define the permutation uniquely is also not difficult to see, when we consider the algorithm constructing the permutation from the inv(i)-values --processing these values in the order of increasing i --: this is an algorithm that 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 modifications of the inv-value correspond to a swap of neighbours: each swap of two neighbours changes exactly one inv(i)-value by one, viz. with $i =$ the larger of the two values swapped. The value of inv(i) is to be increased by one if the swap brings the pair into the wrong order, otherwise it has to be decreased by one.

A feasible sequence of inv-values to be generated is now reasonably obvious: it is the generalization of the Grey-code, For $n = 4$ it would begin

0	0	0	0
0	0	0	1
0	0	0	2
0	0	0	3
0	0	1	3
0	0	1	2
0	0	1	1
0	0	1	0
0	0	2	0
0	0	2	1
0	0	2	2
0	0	2	3
0	1	2	3
0	1	2	2

etc.

The rule is as follows: a number is changeable when it may be increased or decreased by one. It may be increased if the sum of the numbers to its left is even and it has not reached its maximum value; it may be decreased if the sum of the numbers to its left is odd and it has not reached its minimum value zero. At each step, always the right-most changeable number is changed.

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 left-hand neighbour (because this is a smaller one, inv(i) has to be increased) or with its right-hand neighbour (a decrease of inv(i) by one), we have to establish the place c in the array, where the element i is positioned; this is given by

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

The reason for this formula: i is its original position, $inv(i)$ is the number of smaller elements to the right of it; we have to add to it the number of larger elements in front of the section with elements $\leq i$.

In the following program we have given $inv(0)$ --which should be constantly 0-- the funny value -2; this is just the usual, mean, little coding trick, in order to let the search for the right-most changeable number terminate artificially when there is no more such a number. The value "totinv" records the total number of inversions in the array a; the variable "linv" records the sum of the (non-funny) inv-values to the left of $inv(i)$.

```
begin glocon n; privar a, inv, ready, totinv;
   a vir int array := (0, 0) {a initialized with  a(0) = 0};
   inv vir int array := (0, -2) {inv initialized with  inv(0) = - 2};
   do inv.dom ≠ n → a:hiext(inv.dom); inv:hiext(0) od
     {a(0),..., a(n-1) = 0,..., n-1   and  inv(0),...,inv(n-1) = -2, 0,...,
   ready vir bool := false; totinv vir int := 0;
   do non ready →
     begin glocon n; glovar a, inv, ready, totinv;
        privar i, c, linv;   printarray (a);
        i vir int := n - 1; c vir int := 0; linv vir int := 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;
        c:= c + i - inv(i);
        if even(linv) and i ≥ 1 → inv:(i)= inv(i) + 1; totinv:= totinv + 1;
                                   a:swap(c - 1, c)
         [] odd(linv) and i ≥ 1 → inv:(i)= inv(i) - 1; totinv:= totinv - 1;
                                   a:swap(c, c + 1)
         [] i = 0 → ready:= true
        fi
     end
   od
end
```

Plataanstraat 5
NL-4565 NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow