ALGORITHM 71
PERMUTATION

R. R. Coveyou and J. G. Sullivan
Oak Ridge National Laboratory, Oak Ridge, Tenn.

```
procedure PERMUTATION (I, P, N);
  value I, N; integer N; integer array P; boolean I;
  comment This procedure produces all permutations of the
    integers from 0 thru N. Upon entry with I = false the pro-
    cedure initializes itself producing no permutation. Upon each
    successive entry into the procedure with I = true a new
    permutation is stored in P[0] thru P[N]. When the process has
    been exhausted a sentinel is set:
    P[0] : −1,
    N ≥ 0;
  begin
    integer i; own integer array x[0:N];
    if ¬ I then
    begin for i := 0 step 1 until N−1 do x[i] := 0; x[N] := −1;
      go to E end;
    for i := N step −1 until 0 do begin if x[i]≠i then go to A;
      x[i] := 0 end;
    P[0] := −1; go to E;
A:  x[i] := x[i]+1; P[0] := 0;
    for i := 1 step 1 until N do
      begin P[i] := P[i−x[i]]; P[i−x[i]] := i end;
E:  end PERMUTATION
```

CERTIFICATION OF ALGORITHM 71
PERMUTATION (R. R. Coveyou and J. G. Sullivan, *Comm. ACM*, Nov. 1961)

P. J. Brown
University of North Carolina, Chapel Hill, N. C.

PERMUTATION was transliterated into GAT for the Univac 1105 and successfully run for a number of cases.

CERTIFICATION OF ALGORITHM 71
PERMUTATION (R. R. Coveyou and J. G. Sullivan, *Comm. ACM*, Nov. 1961)

J. E. L. Peck and G. F. Schrack
University of Alberta, Calgary, Alberta, Canada

PERMUTATION was translated into Fortran for the IBM 1620 and it performed satisfactorily. The **own integer array** x[0:n] may be shortened to x[1:n], provided corresponding corrections are made in the first two **for** statements.

However, PERMUTE (Algorithm 86) is superior to PERMU TATION in two respects.

(1) PERMUTATION, using storage of order 2n, is designed to permute the specific vector 0, 1, 2, $\cdots$, n − 1 rather than an arbitrary vector. Thus storage of order 3n is required to permute an arbitrary vector. PERMUTE, in contrast, only needs storage of order 2n to permute an arbitrary vector.

(2) PERMUTE is built up from cyclic permutations. The number of permutations actually executed internally (the redundant ones are suppressed) by PERMUTE is asymptotic to (e − 1)n! rather than n!. In spite of this, PERMUTE is distinctly faster (1316 against 2823 seconds for n = 8) than PERMUTATION. If t_n is the time taken for all permutations of a vector with n components, and if $r_n = t_n/nt_{n-1}$, then one would expect r_n to be close to 1. Experiment with small values of n gave the following results for r_n.

n	6	7	8
PERMUTE	0.96	0.99	1.00
PERMUTATION	1.10	1.13	1.12

Is there yet a faster way to do it?

See also: C. Tompkins, "Machine Attacks on Problems whose Variables are Permutations", Proceedings of Symposia in Applied Mathematics, Vol. VI: *Numerical Analysis* (N. Y., McGraw-Hill, 1956).

CERTIFICATION OF ALGORITHM 71
PERMUTATION [R. R. Coveyou and J. G. Sullivan, *Comm. ACM*, Nov. 1961]

J. S. Hillmore
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The algorithm was successfully run using the Elliott Algol translator on the National-Elliott 803. The integer array x was made a parameter of the procedure in order to avoid having an **own** array with variable bounds.