ALGORITHM 87
PERMUTATION GENERATOR

JOHN R. HOWELL
Orlando Aerospace Division, Martin Marietta Corp., Orlando, Florida

```
procedure PERMUTATION (N, K);
value K, N;  integer K;  integer array N;
comment This procedure generates the next permutation in
    lexicographic order from a given permutation of the K marks
    0, 1, ··· , (K−1) by the repeated addition of (K−1) radix K.
    The radix K arithmetic is simulated by the addition of 9 radix
    10 and a test to determine if the sum consists of only the original
    K digits. Before each entry into the procedure the K marks
    are assumed to have been previously specified either by input
    data or as the result of a previous entry. Upon each such entry a
    new permutation is stored in N[1] through N[K]. In case the
    given permutation is (K−1), (K−2), ··· , 1, 0, then the next
    permutation is taken to be 0, 1, ··· , (K − 1). A FORTRAN
    subroutine for the IBM 7090 has been written and tested for
    several examples;
begin integer i, j, carry;
  for i := 1 step 1 until K do
      if N[i] − K + i ≠ 0 then go to add;
  for i := 1 step 1 until K do N[i] := i − 1;
  go to exit;
add: N[K] := N[K] + 9;
      for i := 1 step 1 until K−1 do
          begin if K > 10 then go to B;
                carry := N[K−i+1]÷10;  go to C;
          B:    carry := N[K−i+1]÷K;
          C:    if carry = 0 then go to test;
                N[K−i] := N[K−i] + carry;
                N[K−i+1] := N[K−i+1] −10 × carry
          end i;
test: for i := 1 step 1 until K do if N[i] − (K − 1) > 0
      then go to add;
      for i := 1 step 1 until K−1 do
          for j := i+1 step 1 until K do
              if N[i]−N[j] = 0 then go to add;
exit: end PERMUTATION GENERATOR
```

CERTIFICATION OF ALGORITHM 87
PERMUTATION GENERATOR [John R. Howell, *Comm. ACM*, Apr. 1962]

D. M. COLLISON
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The array N was removed from the value list in order that the permutations might be available outside the procedure. The algorithm was then run successfully with the Elliott ALGOL translator on the National-Elliott 803. It was rather slower than Algorithm 86.

CERTIFICATION OF ALGORITHM 87
PERMUTATION GENERATOR [John R. Howell, *Comm. ACM* (Apr. 1962)]

G. F. SCHRACK and M. SHIMRAT
University of Alberta, Calgary, Alb., Canada

PERMUTATION GENERATOR was translated into FORTRAN for the IBM 1620 and it performed satisfactorily. The algorithm was timed for several small values of n. For purposes of comparison we include the times (in seconds) for PERMULEX (Algorithm 102).

n	3	4	5	6	7
PERMUTATION GENERATOR	3	41	558	—	—
PERMULEX	—	3	6	37	278

As can be seen from this table, PERMUTATION GENERATOR is considerably slower. It is probable that one could speed up PERMUTATION GENERATOR to a great extent by rearranging the algorithm in such a manner that the digits of a number to a certain base are permuted rather than the elements of a sequence.

REMARKS ON:

ALGORITHM 87 [G6]
PERMUTATION GENERATOR
[John R. Howell, *Comm. ACM 5* (Apr. 1962), 209]
ALGORITHM 102 [G6]
PERMUTATION IN LEXICOGRAPHICAL ORDER
[G. F. Schrak and M. Shimrat, *Comm. ACM 5* (June (1962), 346]
ALGORITHM 130 [G6]
PERMUTE
[Lt. B. C. Eaves, *Comm. ACM 5* (Nov. 1962), 551]
ALGORITHM 202 [G6]
GENERATION OF PERMUTATIONS IN LEXICOGRAPHICAL ORDER
[Mok-Kong Shen, *Comm. ACM 6* (Sept. 1963), 517]

R. J. ORD-SMITH (Recd. 11 Nov. 1966, 28 Dec. 1966 and 17 Mar. 1967)
Computing Laboratory, University of Bradford, England

A comparison of the published algorithms which seek to generate successive permutations in lexicographic order shows that Algorithm 202 is the most efficient. Since, however, it is more than twice as slow as transposition Algorithm 115 [H. F. Trotter, Perm, *Comm. ACM 5* (Aug. 1962), 434], there appears to be room for improvement. Theoretically a "best" lexicographic algorithm should be about one and a half times slower than Algorithm 115. See Algorithm 308 [R. J. Ord-Smith, Generation of Permutations in Pseudo-Lexicographic Order, *Comm. ACM 10* (July 1967), 452] which is twice as fast as Algorithm 202.

ALGORITHM 87 is very slow.

ALGORITHM 102 shows a marked improvement.

ALGORITHM 130 does not appear to have been certified before. We find that, certainly for some forms of vector to be permuted, the algorithm can fail. The reason is as follows.

At execution of $A[f] := r$; on line prior to that labeled *schell*, f has not necessarily been assigned a value. f has a value if, and only if, the Boolean expression $B[k] > 0 \wedge B[k] < B[m]$ is **true** for at least one of the relevant values of k. In particular when matrix A is set up by $A[i] := i$; for each i the Boolean expression above is **false** on the first call.

ALGORITHM 202 is the best and fastest algorithm of the exicographic set so far published.

A collected comparison of these algorithms is given in Table I. t_n is the time for complete generation of $n!$ permutations. Times are scaled relative to t_8 for Algorithm 202, which is set at 100. Tests were made on an ICT 1905 computer. The actual time t_8 for Algorithm 202 on this machine *was* 100 seconds. r_n has the usual definition $r_n = t_n/(n \cdot t_{n-1})$.

TABLE I

Algorithm	t_6	t_7	t_8	r_6	r_7	r_8
87	118	—	—	—	—	—
102	2.1	15.5	135	1.03	1.08	1.1
130	—	—	—	—	—	—
202	1.7	12.4	100	1.00	1.00	1.00