```
      begin integer km;  real t;
        t := x[1];  km := k - 1;
        for i := 1 step 1 until km do
            x[i] := x[i+1];
        x[k] := t;  p[k] := p[k] - 1;
        if p[k] ≠ 0 then go to EXIT;
        p[k] := k
      end k;
  first := true;
EXIT: end PERMUTE
```

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 35
SIEVE (T. C. Wood, *Comm. ACM*, March 1961)

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

SIEVE was transliterated into GAT for the UNIVAC 1105 and successfully run for a number of cases.

The statement:

go to if n/p[i] = n ÷ p[i] **then** b1 **else** b2;

was changed to the statement:

go to if n/p[i] − n ÷ p[i] < .5/Nmax **then** b1 **else** b2;

Roundoff error might lead to the former giving undesired results.

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 PERMUTATION in two respects.

(1) PERMUTATION, using storage of order 2n, is designed to permute the specific vector 0, 1, 2, ···, 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).

The Calculation of Easter...

By Donald Knuth
California Institute of Technology, Pasadena, California

Here are two programs, written to demonstrate ALGOL and COBOL. Object: to determine the month and day of Easter, given the year. The ALGOL program (1) is written as a procedure, which sets up "month" and "day" given the value of "year." The COBOL program (2) prepares a printed table of Easter date, from 500 to 4999 A.D.

(1)

```
procedure Easter (year, month, day); value year; integer
    year, month, day;
comment This procedure calculates the day and month of
    Easter given the year. It gives the actual date of "Western
    Easter" (not the Eastern Easter of the Eastern Orthodox
    churches) after A.D. 463. "golden number" is the number of the
    year in the Metonic cycle, used to determine the position of the
    calendar moon. "Gregorian correction" is the number of preced-
    ing years like 1700, 1800, 1900 when leap year was not held,
    "Clavian correction" is a correction for the Metonic cycle of about
```

```
  8 days every 2500 years. "epact" is the age of the calendar moon
  at the begnning of the year. "extra days" specifies when Sunday
  occurs in March. "epact" specifies when full moon occurs. Easter
  is the first Sunday following the first full moon which occurs
  on or after March 21. Reference: A. De Morgan, A Budget of
  Paradoxes;
begin integer golden number, century, Gregorian correction,
  Clavian correction, extra days, epact;
integer procedure mod (a, b); value a, b; integer a, b;
  mod := a − b × (a ÷ b);
golden number := mod (year, 19) + 1; if year ≦ 1582 then go
  to Julian;
Gregorian: century := year ÷ 100 + 1;
  Gregorian correction := (3 × century) ÷ 4 − 12;
  Clavian correction := (century − 16 − (century − 18) ÷ 25)
    ÷ 3;
  extra days := (5 × year) ÷ 4 − Gregorian correction − 10;
  epact := mod (11 × golden number + 20 + Clavian correc-
    tion − Gregorian correction, 30);
  if epact ≦ 0 then epact := epact + 30;
  if (epact = 25 ∧ golden number > 11) ∨ epact = 24 then
    epact := epact + 1;
  go to ending routine;
Julian: extra days := (5 × year) ÷ 4; epact := mod (11 ×
  golden number −4, 30) + 1;
ending routine: day := 44 − epact; if day < 21 then day :=
    day + 30;
  day := day + 7 − mod (extra days + day, 7);
  if day > 31 then begin month := 4; day := day − 31 end
    else month := 3 end Easter
```

(2)

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. DATE OF EASTER.
000300 AUTHOR. D E KNUTH.
000400 DATE-WRITTEN. JANUARY 22, 1962.
000500 DATE-COMPILED. JANUARY 23, 1962.
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SOURCE-COMPUTER. COBOLIAC.
000900 OBJECT-COMPUTER. COBOLIAC-2, PRINTER.
001000 SPECIAL-NAMES.
001100     PRINTER-OVERFLOW IS SKIP-TO-NEXT-PAGE.
001200 INPUT-OUTPUT SECTION.
001300 FILE-CONTROL.
001400     SELECT ANSWER-TABLE, ASSIGN TO PRINTER.
001500 DATA DIVISION.
001600 FILE SECTION.
001700 FD  ANSWER-TABLE; LABEL RECORDS ARE STANDARD; DATA
               RECORD IS EASTER-DATES.
001800     01  EASTER-DATES.
001900         02  EASTER-DAY; OCCURS 6 TIMES.
002000             03  MONTH; SIZE IS 5 ALPHABETIC DISPLAY
                       CHARACTERS.
002100             03  FILLER; SIZE IS 1 CHARACTERS.
002200             03  DAYS; PICTURE IS Z9,.
002300             03  YEARS; PICTURE IS ZZ999.
002400             03  FILLER; SIZE IS 6 CHARACTERS.
002500 WORKING-STORAGE SECTION.
002600 77  TEMP; SIZE 6 NUMERIC COMPUTATIONAL.
002700 77  TEMP-1; SIZE 6 NUMERIC COMPUTATIONAL.
002800 77  BASE-YEAR; SIZE 4 NUMERIC COMPUTATIONAL.
002900 77  LINE; SIZE 2 NUMERIC COMPUTATIONAL.
003000 77  COLUMN; SIZE 1 NUMERIC COMPUTATIONAL.
003100 77  COLUMN-YEAR; SIZE 4 NUMERIC COMPUTATIONAL.
003200 77  YEAR; SIZE 4 NUMERIC COMPUTATIONAL.
003300 77  GOLDEN-NUMBER; SIZE 2 NUMERIC COMPUTATIONAL.
003400 77  CENTURY; SIZE 2 NUMERIC COMPUTATIONAL.
003500 77  GREGORIAN-CORRECTION; SIZE 2 NUMERIC
               COMPUTATIONAL.
003600 77  CLAVIAN-CORRECTION; SIZE 2 NUMERIC
               COMPUTATIONAL.
003700 77  EXTRA-DAYS; SIZE 4 NUMERIC COMPUTATIONAL.
003800 77  EPACT; SIZE 2 NUMERIC COMPUTATIONAL.
003900 77  DAY; SIZE 2 NUMERIC COMPUTATIONAL.
004000 PROCEDURE DIVISION.
004001 CONTROL SECTION.
004002 OUTER-LOOP.
004100     OPEN OUTPUT ANSWER-TABLE.
004200     PERFORM MIDDLE-LOOP VARYING BASE-YEAR FROM 500
               BY 300
004300     UNTIL BASE-YEAR EQUALS 5000.
004400     STOP RUN.
004500 MIDDLE-LOOP.
004600     PERFORM INNER-LOOP VARYING LINE FROM 0 BY 1
               UNTIL LINE EQUALS 50.
004700 INNER-LOOP.
004800     PERFORM COMPUTATION VARYING COLUMN FROM 1
               BY 1 UNTIL COLUMN
004900     EXCEEDS 6. IF LINE IS NOT EQUAL TO 49, WRITE
               EASTER-DATES;
005000     OTHERWISE WRITE EASTER-DATES BEFORE
               SKIP-TO-NEXT-PAGE.
005100 COMPUTATION SECTION.
005200 FIND-YEAR.
005300     MULTIPLY COLUMN BY 50 GIVING COLUMN-YEAR; ADD
               COLUMN-YEAR,
005400     BASE-YEAR, LINE, AND −50 GIVING YEAR.
005500 FIND-GOLDEN-NUMBER.
005600     DIVIDE 19 INTO YEAR GIVING TEMP; MULTIPLY 19 BY
               TEMP;
005700     SUBTRACT TEMP FROM YEAR GIVING
               GOLDEN-NUMBER THEN
005800     ADD 1 TO GOLDEN-NUMBER.
005900     IF YEAR IS LESS THAN 1583 GO TO JULIAN.
006000 GREGORIAN.
006100     DIVIDE 100 INTO YEAR GIVING CENTURY; ADD 1 TO
               CENTURY.
006200     MULTIPLY CENTURY BY 3 GIVING TEMP; DIVIDE 4
               INTO TEMP;
006300     SUBTRACT 12 FROM TEMP GIVING
               GREGORIAN-CORRECTION.
006400     SUBTRACT 18 FROM CENTURY GIVING TEMP; DIVIDE
               25 INTO TEMP;
006500     SUBTRACT TEMP AND 16 FROM CENTURY GIVING
               TEMP;
006600     DIVIDE 3 INTO TEMP GIVING CLAVIAN-CORRECTION.
006700     MULTIPLY YEAR BY 5 GIVING TEMP; DIVIDE 4 INTO
               TEMP;
006800     SUBTRACT 10 AND GREGORIAN-CORRECTION FROM
               TEMP GIVING EXTRA-DAYS.
006900 FUDGE-EPACT.
007000     MULTIPLY GOLDEN-NUMBER BY 11 GIVING TEMP;
               SUBTRACT
007100     GREGORIAN-CORRECTION FROM TEMP; ADD 19,
               CLAVIAN-CORRECTION, TEMP;
007200     DIVIDE 30 INTO TEMP GIVING TEMP-1; MULTIPLY 30
               BY TEMP-1;
007300     SUBTRACT TEMP-1 FROM TEMP; ADD TEMP, 1 GIVING
               EPACT.
007400     IF EPACT EQUALS 24 OR (25 AND GOLDEN-NUMBER IS
               GREATER THAN 11)
007500     ADD 1 TO EPACT.
007600     GO TO ENDING-ROUTINE.
007700 JULIAN.
007800     MULTIPLY YEAR BY 5 GIVING TEMP; DIVIDE 4 INTO
               TEMP GIVING EXTRA-DAYS.
007900     MULTIPLY GOLDEN-NUMBER BY 11 GIVING TEMP;
               SUBTRACT 4 FROM TEMP;
008000     DIVIDE 30 INTO TEMP GIVING TEMP-1; MULTIPLY 30
               BY TEMP-1;
008100     SUBTRACT TEMP-1 FROM TEMP; ADD TEMP AND 1
               GIVING EPACT.
008200 ENDING-ROUTINE.
008300     SUBTRACT EPACT FROM 44 GIVING DAY; IF DAY IS
               LESS THAN 21 ADD
008400     30 TO DAY.
008500 MAKE-DAY-SUNDAY.
008600     ADD DAY, EXTRA-DAYS GIVING TEMP; DIVIDE 7 INTO
               TEMP GIVING TEMP-1;
008700     MULTIPLY 7 BY TEMP-1; SUBTRACT TEMP-1 FROM TEMP;
008800     SUBTRACT TEMP FROM 7 GIVING TEMP; ADD TEMP TO
               DAY.
008900 TRANSFER-ANSWER.
009000     IF DAY EXCEEDS 31 THEN SUBTRACT 31 FROM DAY;
009100     MOVE "APRIL" TO MONTH(COLUMN); OTHERWISE
               MOVE "MARCH" TO MONTH(COLUMN).
009200     MOVE DAY TO DAYS(COLUMN); MOVE YEAR TO
               YEARS(COLUMN).
```

Note. Each line of the COBOL algorithm above which has a blank sequence number represents a continuation of the preceding line. In the standard COBOL reference format these two lines would actually be punched onto a single card. They are broken into two parts here for typographical reasons only.