Algorithms

G. E. FORSYTHE, J. G. HERRIOTT, Editors

ALGORITHM 242
PERMUTATIONS OF A SET WITH REPETITIONS [G6]

T. W. Sag (Recd. 10 Feb. 1964 and 19 June 1964)
Math. Dept., Manchester U., Manchester, England

```
procedure PERMUTATION (X, K, j, process);
  array X; integer array K; integer j; procedure process;
comment PERMUTATION generates all the distinct permuta-
  tions of an array of numbers consisting of K[1] numbers equal to
  X[1], K[2] numbers equal to X[2], ··· , K[j] numbers equal to
  X[j]. The K[i]'s must be positive integers. Each permutation is
  stored in the array Y and processed according to the user's wish
  by the procedure process before the next permutation is gen-
  erated.
    {The procedure is more efficient if the sequence K[i] is mono-
  tone decreasing.—Ref.};
begin
  real x; integer M, N, i; array B[1:K[j]];
  procedure permutation (x, M, N, j, B, process);
  real x; integer M, N, j; array B; procedure process;
  begin
        real A; integer i, KK, N1, N2, j1;
          integer array J[1:N+1];
        array Y[1:N+M]; N2 := N + M;
        if M = 0 then go to 1;
        for i := N + 1 step 1 until N2 do Y[i] := x;
    1:  for i := 1 step 1 until N do J[i] := i;
        J[N+1] := N2 + 1; j1 := j − 1; KK := N;
    2:  for i := 1 step 1 until KK do Y[J[i]] := B[i];
        if j1 ≦ 1 then begin process(Y); go to 3 end;
        A := X[j1−1]; N1 := K[j1−1];
        permutation (A, N1, N2, j1, Y, process);
    3:  for i := 1 step 1 until N do
        begin
              Y[J[i]] := x; J[i] := J[i] + 1;
              if J[i] − J[i+1] + 1 ≦ 0 then go to 4 else go to 5;
          4:  KK := i; go to 2;
          5:  J[i] := i
        end
  end of permutation;
  if j = 1 then begin x := X[1]; M := 0; go to 1 end;
  x := X[j−1]; M := K[j−1];
1:  for i := 1 step 1 until K[j] do B[i] := X[j];
    permutation (x, M, K[j], j, B, process);
end of PERMUTATION
```

CERTIFICATION OF ALGORITHM 203 [E4]
STEEP1 [E. J. Wasscher, *Comm. ACM 9* (Sept. 1963), 517]

Philip Wallack (Recd. 25 May 1964)
Republic Aviation Corp., Farmingdale, L. I., N. Y.

*STEEP*1 was translated into Fortran IV and run on the IBM 7094. The program was tested on the function $x^4 + y^4 - 1$, with starting values $x = y = 1.5$. Other parameter values were those suggested in the body of the algorithm. After 17 steps the values of the variables were $x = .0180$, $y = .0191$, and the function value $fmin = -.9999999$.

I feel that good programming practice requires that a count be kept of the number of steps taken in *STEEP*1 and the number of iterations in *ATIVE*, with running checks on both these quantities to control looping. Counters were set up for this purpose in the version of the program I ran.

CERTIFICATION OF ALGORITHM 207 [M1]
STRINGSORT [J. Boothroyd, *Comm. ACM 6* (Oct. 1963), 615]

Charles R. Blair (Recd. 31 Jul. 1964)
Department of Defense, Washington 25, D. C.

STRINGSORT compiled and ran successfully without correction on the Aldap translator for the CDC 1604A. The following sorting times were observed.

Number of Items	*Time in Seconds*
10	0.03
20	0.05
50	0.20
100	0.38
200	1.03
500	3.22
1000	6.43
2000	12.85
5000	38.72
10000	90.72

CERTIFICATION OF ALGORITHM 218 [D2]
KUTTA MERSON [Phyllis M. Lukehart, *Comm. ACM 6* (Dec. 1963), 737]

Karen Borman Priebe (Recd. 10 Feb. 1964)
Woodward Governor Company, Rockford, Illinois

Algorithm 218 was translated into Fast for the NCR 315 and gave satisfactory results with the following corrections, if the equations were scaled as recommended in the comment of the original algorithm. Ignoring this scaling can lead to results that do not satisfy the intended error criterion.

1. **procedure** *KuttaMerson (n, t, y, eps, h, fct, first, x)*;
instead of
procedure *KuttaMerson (n, t, y, eps, h, fct, first)*;
2. **real array** *y, x*;
instead of
real array *y*;
3. **if** *first* **then begin for** *i* := 1 **step** 1 **until** *n* **do** *y*0[*i*] := *y*[*i*];
hc := *h*;
instead of
if *first* **then begin** *hc* := *h*; ···

4. **if** *loc* < *ploc* **then**
begin
if *increase* /\ *loc* = (*loc*÷2) × 2 /\ *ploc* > 1 **then**
begin
hc := 2 × *hc*;
loc := *loc* ÷ 2;
ploc := *ploc* ÷ 2
end;
go to *next*
end;
for *i* := 1 **step** 1 **until** *n* **do** *x*[*i*] := *y*0[*i*];
end *KuttaMerson*

instead of

if *loc* < *ploc* /\ *increase* ···
end *KuttaMerson*

5. The following sentences should be added to the initial comment of the procedure:

The values of the dependent variables at $t + h$ are placed in the array x. Note that the values of t and *first* are changed as side-effects of the procedure. {As originally written, *KuttaMerson* seemed unable to obtain the values of the solution at t or to transmit the values of the solution at $t + h$ to the outside program!—Ed.}

6. Change *array* to **array** in the body of the procedure.

7. Insert after **own integer** *ploc*;

own array *y*0[1:*n*];

Delete *y*0 from the existing array declaration.

Revised Algorithms Policy • May, 1964

A contribution to the Algorithms department must be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double-spaced in capital and lower-case letters. Authors should carefully follow the style of this department, with especial attention to indentation and completeness of references. Material to appear in **boldface** type should be underlined in black. Blue underlining may be used to indicate *italic* type, but this is usually best left to the Editor.

An algorithm must be written in the ALGOL 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1-17], and normally consists of a commented procedure declaration. Each algorithm must be accompanied by a complete driver program in ALGOL 60 which generates test data, calls the procedure, and outputs test answers. Moreover, selected previously obtained test answers should be given in comments in either the driver program or the algorithm. The driver program may be published with the algorithm if it would be of major assistance to a user.

Input and output should be achieved by procedure statements, using one of the following five procedures (whose body is not specified in ALGOL): [see "Report on Input-Output Procedures for ALGOL 60," *Comm, ACM 7* (Oct. 1964), 628-629].

procedure *inreal* (*channel, destination*)· **value** *channel*; **integer** *channel*; **real** *destination*; **comment** the number read from channel *channel* is assigned to the variable *destination*; . . . ;
procedure *outreal* (*channel, source*); **value** *channel, source*; **integer** *channel*; **real** *source*; **comment** the value of expression *source* is output to channel *channel*; . . . ;
procedure *ininteger* (*channel, destination*);
value *channel*; **integer** *channel, destination*; . . . ;
procedure *outinteger* (*channel, source*);
value *channel, source*; **integer** *channel, source*; . . . ;
procedure *outstring* (*channel, string*); **value** *channel*; **integer** *channel*; **string** *string*; . . . ;

If only one channel is used by the program, it should be designated by 1. Examples:

outstring (1, '*x* ='); *outreal* (1, *x*);
for *i* := 1 **step** 1 **until** *n* **do** *outreal* (1, *A*[*i*]);
ininteger (1, *digit* [17]);

It is intended that each published algorithm be a well-organized, clearly commented, syntactically correct, and a substantial contribution to the ALGOL literature. All contributions will be refereed both by human beings and by an ALGOL compiler. Authors should give great attention to the correctness of their programs, since referees cannot be expected to debug them. Because ALGOL compilers are often incomplete, authors are encouraged to indicate in comments whether their algorithms are written in a recognized subset of ALGOL 60 [see "Report on SUBSET ALGOL 60 (IFIP)," *Comm. ACM 7* (Oct, 1964), 626-627].

Certifications and remarks should add new information to that already published. Readers are especially encouraged to test and certify previously uncertified algorithms. Rewritten versions of previously published algorithms will be refereed as new contributions, and should not be imbedded in certifications or remarks.

Galley proofs will be sent to the authors; obviously rapid and careful proofreading is of paramount importance.

Although each algorithm has been tested by its author, no liability is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.—G.E.F.

CERTIFICATION OF ALGORITHM 221 [S14] GAMMA FUNCTION [Walter Gautschi, *Comm. ACM 7* (Mar. 1964), 143]

VAN K. MCCOMBS (Recd. 10 Apr. 1964 and 1 Jun. 1964)
General Electric Co., Huntsville, Ala.

The algorithm was translated into FORTRAN IV for the IBM 7094. Computations were performed in double precision to take advantage of the ten significant digits given by the polynomial coefficients. The function $\Gamma(z)$ was evaluated for the range $0 < z \leqq 10$ with an increment of 0.1, and the results were checked with the values published in Table of the Gamma Function for Complex Arguments, *NBS Applied Mathematics Series 34* (1954). The algorithm gave ten-digit accuracy for the range indicated.

CERTIFICATION OF ALGORITHM 225 [S14] GAMMA FUNCTION WITH CONTROLLED ACCURACY [S. J. Cyvin and B. N. Cyvin, *Comm. ACM 7* (May 1964), 295]

T. A. BRAY (Recd. 25 May 1964 and 18 Jun. 1964)
Boeing Scientific Research Laboratories, Seattle, Wash.

Algorithm 225 was coded in FORTRAN II and run on the IBM 1620. No corrections were necessary and the following results were obtained for $m = 2$:

x	*GAMMA* (*m, x*)	*x*	*GAMMA* (*m, x*)
.01	99.44362100	3.50	3.32349920
.05	19.47214000	4.00	6.00067550
.10	9.51444650	4.50	11.63224700
.50	1.77253280	5.00	24.00270200
1.00	1.00011250	5.50	52.34511500
1.50	.88626644	10.00	0.36286974_{10} 6
2.00	1.00011250	25.00	0.62043066_{10} 24
2.50	1.32939960	50.00	0.60826434_{10} 63
3.00	2.00022510		

These results are correct to at least two significant digits. The following results and times were obtained for $x = 0.5$:

m	*GAMMA* (*m, x*)	*TIME* (*in seconds*)
2	1.77253280	58
3	1.77254230	105
4	1.77245370	200
5	1.77244430	405
6	1.77244020	885

The correct result is 1.7724539. Note that the accuracy decreased as m increased and the result for $m = 6$ is incorrect in the sixth significant digit.

This algorithm is extremely slow as compared to some others available. Algorithm 31 was used for the above set of arguments and gave seven-digit accuracy in 250 milliseconds per argument.