Algorithms

G. E. FORSYTHE, J. G. HERRIOT, Editors

ALGORITHM 248
NETFLOW [H]

WILLIAM A. BRIGGS (Recd. 18 Jan. 1964 and 17 Aug. 1964)
Marathon Oil Company, Findlay, Ohio

```
procedure NETFLOW (nodes, arcs, I, J, cost, hi, lo, flow, pi,
    INFEAS);
  value nodes, arcs;  integer nodes, arcs;
  integer array I, J, cost, hi, lo, flow, pi;  label INFEAS;
comment This procedure determines the least-cost flow pat-
  tern over an upper and lower bound capacitated flow network.
    Each directed network arc a is defined by nodes I[a] and
  J[a], has upper and lower flow bounds hi[a] and lo[a], and cost
  per unit of flow cost[a]. Costs and flow bounds may be any
  positive or negative integers. An upper flow bound must be
  greater than or equal to its corresponding lower flow bound for
  a feasible solution to exist. There may be any number of parallel
  arcs connecting any two nodes.
    A multi-source, multi-demand, capacitated transportation or
  transshipment problem may be stated as a network flow problem
  as follows:
    Append to the network (1) bounded arcs from the demand
  node(s) to a "super sink," (2) bounded arcs from a "super
  source" to the supply node(s), (3) an arc directed from the
  "super sink" to the "super source" with zero lower bound, a
  large positive upper bound, and a negatively large cost.
    NETFLOW will maximize flow through the low-cost arc from
  "supper sink" to "super source"—subject to the capacity
  constraints of the network—fulfilling all demands optimally.
    The procedure returns vectors flow and pi. Flow[a] is the
  computed optimal flow over network arc a. Pi[n] is a number
  —the dual variable—which represents the relative value of
  injecting one unit of flow into the network at node n. NETFLOW
  may be entered with any values in vectors flow and pi (such as
  those from a previous or a guessed solution) feasible or not. If
  the initial contents of flow do not conserve flow at any node,
  the solution values will also not conserve flow at that node, by
  the same amount. This fact can be frequently used to advantage
  in transportation problem definition. The closer initial values
  of flow and pi are to solution values, the shorter the computa-
  tion.
    Procedure NETFLOW is a mechanization of the out-of-kilter
  network flow algorithm described by D. R. FULKERSON in J.
  Soc. Indust. Appl. Math. 9 (1961), 18–27, and elsewhere. Many
  thanks are due the referee for noting some erroneous comments
  and for suggesting ways to increase the efficiency and utility of
  the procedure;
begin integer a, aok, c, cok, del, e, eps, inf, lab, n, ni, nj, src, snk;
  integer array na, nb[1: nodes];
  integer procedure min (x, y); value x, y; integer x, y;
    begin if x < y then min := x else min := y end min;
  comment check feasibility of formulation;
  for a := 1 step 1 until arcs do if lo[a] > hi[a] then go to INFEAS;
  inf := 99999999; comment set inf to max available integer;
  aok := 0;
  comment find an out-of-kilter arc;
Seek: for a := 1 step 1 until arcs do
  begin c := cost [a] + pi [I[a]] − pi [J[a]];
    if flow [a] < lo [a] ∨ (c<0∧ flow[a]<hi[a]) then
      begin src := J [a]; snk := I [a]; e := +1; go to LABL
      end;
    if flow [a] > hi [a] ∨ (c>0∧flow[a]>lo[a]) then
      begin src := I[a]; snk := J[a]; e := −1; go to LABL
      end;
  end;
  comment no remaining out-of-kilter arcs;
  go to FINI;
  comment attempt to bring found out-of-kilter arc into kilter;
LABL: if a = aok ∧ na[src] ≠ 0 then go to SKIP;
  aok := a;
  for n := 1 step 1 until nodes do na[n] := nb[n] := 0;
  na[src] := abs (snk) × e; nb[src] := abs (aok) × e;
SKIP: cok := c;
LOOP: lab := 0;
  for a := 1 step 1 until arcs do
  begin if (na[I[a]]=0∧na[J[a]]=0) ∨
      (na[I[a]]≠0∧na[J[a]]≠0) then go to XC;
    e := cost[a] + pi[I[a]] − pi[J[a]];
    if na[I[a]] = 0 then go to XA;
    if flow[a] ≧ hi[a] ∨ (flow[a]≧lo[a]∧c>0) then go to XC;
    na[J[a]] := I[a]; nb[J[a]] := a; go to XB;
XA: if flow[a] ≦ lo[a] ∨ (flow[a]≦hi[a]∧c<0) then go to XC;
    na[I[a]] := −J[a]; nb[I[a]] := −a;
XB: lab := +1;
    comment node labeled, test for breakthru;
    if na[snk] ≠ 0 then go to INCR;
XC: end;
  comment no breakthru;
  if lab ≠ 0 then go to LOOP;
  comment determine change to pi vector;
  del := inf;
  for a := 1 step 1 until arcs do
  begin if (na[I[a]]=0∧na[J[a]]=0) ∨
      (na[I[a]]≠0∧na[J[a]]≠0) then go to XD;
    e := cost[a] + pi[I[a]] − pi[J[a]];
    if na[J[a]] = 0 ∧ flow[a] < hi[a] then del := min (del, e);
    if na[J[a]] ≠ 0 ∧ flow[a] > lo[a] then del := min (del, −e);
XD: end;
  if del = inf ∧ (flow[aok]=hi[aok]∨flow[aok]=lo[aok]) then
    del := abs (cok);
  if del = inf then go to INFEAS; comment exit, no feasible
  flow pattern;
  comment change pi vector by computed del;
  for n := 1 step 1 until nodes do if na[n] = 0 then pi[n] :=
    pi[n] + del;
  comment find another out-of-kilter arc;
  go to SEEK;
  comment breakthru, compute incremental flow;
INCR: eps := inf;
  ni := src;
BACK: nj := abs (na[ni]); a := abs (nb[ni]);
  c := cost[a] − abs (pi[ni]−pi[nj]) × sign (nb[ni]);
  if nb[ni] < 0 then go to XE;
  if c > 0 ∧ flow[a] < lo[a] then eps := min (eps, lo[a]−flow[a]);
```

```
    if c ≦ 0 ∧ flow[a] < hi[a] then eps := min (eps, hi[a]−flow[a]);
    go to XF;
  XE: if c < 0 ∧ flow[a] > hi[a] then eps := min (eps,flow[a]
    −hi[a]);
    if c ≧ 0 ∧ flow[a] > lo[a] then eps := min (eps, flow[a]−lo[a]);
  XF: ni := nj; if ni ≠ src then go to BACK;
    comment change flow vector by computed eps;
  BACK2: nj := abs (na[ni]); a := abs (nb[ni]);
    flow[a] := flow[a] + eps × sign (nb[ni]);
    ni := nj; if ni ≠ src then go to BACK2;
    comment find another out-of-kilter arc;
    aok := 0; go to SEEK;
  FINI: end NETFLOW with a feasible, optimal flow pattern
```

ALGORITHM 249
OUTREAL N [I5]

NIKLAUS E. WIRTH (Recd. 28 Aug. 1964 and 2 Nov. 1964)
Computer Science Div., Stanford U., Stanford, Calif.

```
procedure outreal n (ch,x,n);
  value ch, x, n; real x; integer ch, n;
comment outreal n outputs to channel ch the real number x as
  a sequence of characters with n significant decimal digits in the
  form ± d.d ··· d10 ± d ·· d, where d stands for a digit. Like the
  procedures outboolean, outstring, ininteger (cf. Report on Input-
  Output Procedures for ALGOL 60, [Comm. ACM 7, (Oct. 1964),
  628-629]) and inreal [Alg. 239, Comm.ACM 7 (Aug.1964), 481] it
  constitutes an example of the use of the primitive procedure
  pair insymbol-outsymbol defined in the referenced Report;
begin integer i, j, k, s; real f; integer array a[1:n];
  procedure outchar(x) value x; integer x;
    outsymbol (ch,'0123456789+−.10', x+1);
  s := k := 0; f := ; 1;
  outchar (if x ≧ 0 then 10 else 11); x := abs(x);
  if x = 0 then begin outchar(0); go to L4 end;
  if x ≧ 1 then
    begin L1: f := f × 10; s := s + 1; if x ≧ f then go to L1;
      f := f × 0.1; s := s − 1
    end
else
    begin L2: f := f × 0.1; s := s − 1;
      if x < f then go to L2
    end;
  x := x/f; comment now 1 ≦ x < 10;
  for j := 1 step 1 until n − 1 do
  begin i := entier(x); a[j] := i; x := (x−i) × 10 end;
  a[n] := x;
  for j := n − 1 step −1 until 1 do
  begin if a[j+1] < 10 then go to L6; a[j+1] := 0;
    a[j] := a[j] + 1
  end;
  if a[1] = 10 then begin a[1] := 1; s := s + 1 end;
L6: outchar(a[1]); outchar(12);
  for j := 2 step 1 until n do outchar(a[j]);
  comment now process the scale factor s;
  if s = 0 then go to L4;
  outchar(13);
  outchar (if s ≧ 0 then 10 else 11); s := abs(s);
  j := 10;
L3: if s ≧ j then begin j := j × 10; k := k + 1; go to L3 end;
L5: if k > 0 then
  begin j := j ÷ 10; i := s ÷ j; outchar (i); s := s − i × j;
    k := k − 1; go to L5
  end;
  outchar(s);
L4:
end
```

ALGORITHM 250
INVERSE PERMUTATION [G6]

B. H. BOONSTRA (Recd. 12 Oct. 1964)
Nationaal Kasregisters, NCR Holland, Amsterdam.

```
procedure inversepermutation (P) of natural numbers up to: (n);
  value n; integer n; integer array P;
comment given a permutation P(i) of the numbers i = 1(1)n,
  the inverse permutation is computed in situ. The process is
  based on the lemma that any permutation can be written as a
  product of mutually exclusive cycles. Procedure inversepermuta-
  tion has been tested for several permutations including n = 1;
begin integer i, j, k, first;
  switch sss := tag, cycle, next, endcycle, finish;
tag: for i := 1 step 1 until n do P[i] := −P[i];
  comment now P[i] contains a negative number if original and
    a positive number if inverse;
  first := 1;
cycle: k := first; i := −P[k];
next: j := −P[i]; P[i] := k;
  if i = first then go to endcycle;
  k := i; i := j; go to next;
endcycle: if first = n then go to finish;
  first := first + 1;
  if P[first] < 0 then go to cycle else go to endcycle;
finish: end inversepermutation
```

REMARK ON ALGORITHM 135 [F4]
CROUT WITH EQUILIBRATION AND ITERATION

[W. M. McKeeman, *Comm. ACM 5* (Nov. 1962), 553–555, 557; 7 (July 1964), 421]

LOREN P. MEISSNER (Recd. 21 Oct. 1964)
Lawrence Radiation Lab., U. of California, Berkeley.

1. The following error in the published algorithm is noted: The procedure *IP*1 forms the sum of $A[i, p] \times A[p, k]$; however, two lines above the bottom line of procedure *CROUT* an attempt is made to use *IP*1 to form the sum of $A[k, p] \times A[p, j]$.

A possible way of correcting this is to add a procedure *IP*1*a* which is identical with *IP*1 except that k is written for i and j for k. Since the procedure is used often, making the correction in this way is not unreasonable. A more extensive undertaking would be to modify *CROUT* to use a more general procedure such as *INNERPRODUCT* [1].

2. The following comment is made in view of the reference to this algorithm in a recent Editor's Note [2]: In the use of Algorithm 135 as a determinant evaluator, it may be well to set m, the "number of right-hand sides" to 1 instead of zero and give an arbitrary nonzero right-hand side such as (1, 0, 0, ···). This will cause a calculation of the "condition," and possibly an exit to *singular*, to call the user's attention to cases in which the determinant is nonsense.

REFERENCES:
1. FORSYTHE, G. E. Crout with Pivoting. Algorithm 16. *Comm. ACM 3* (Sept. 1960), 507.
2. ROTENBERG, L. J. Remark on Revision of Algorithm 41. *Comm. ACM 7* (Mar. 1964), 144.

REMARK ON ALGORITHM 206 [B1]
ARCCOSSIN [Misako Konda, *Comm. ACM 6* (Sept. 1963), 519]

HENRY J. BOWLDEN (Recd. 30 Sept. 1964 and 5 Nov. 1964)
Westinghouse Electric Corp., R&D Ctr., Pittsburgh, Pa.

Algorith ı 206 was transcribed into Burroughs Extended ALGOL after correcting one typographical error, namely the value of $\pi/2$ in the statement labeled *L*3, which should be 1.5707963.

Results were obtained for a selection of values of the argument between 0 and 1. Accuracy is about 7+ decimal digits over the entire range, by comparison with the tables of inverse sines in [*Handbook of Mathematical Functions*, National Bureau of Standards Applied Mathematics Series #55, U.S. Government Printing Off., Washington, D.C., June 1964, 203-212]. Average execution time was 43 milliseconds.

The efficiency of the procedure could be significantly improved by avoiding the computation of $a \times 2 \uparrow (-r-1)$. Powers of 0.5 may be accumulated within the loop, and the modification of A may be skipped entirely when $a = 0$. Actually, if efficiency is important, procedures using the intrinsic *arctan* and the common trigonometric identities are preferable. Such routines, on the B-5000, give full machine accuracy (11+ significant figures) in about 2 milliseconds execution time.

CERTIFICATION OF ALGORITHM 234 [S23] POISSON-CHARLIER POLYNOMIALS [J. M. S. Simões-Pereira. *Comm. ACM 7* (July 1964), 420]

P. A. Samet (Recd. 17 Aug. 1964)
Computation Lab., The University, Southampton, Eng.

PC polynomial was compiled correctly by the Pegasus-Algol compiler and ran without trouble. The procedure was tested for $n = 0(1)4$, values of a in the range 0.2 to 2.0, and x in the range 0 to 1. The values produced were spotchecked by hand.

The procedure could be improved by

(i) putting x, n, a in the **value** part.

(ii) replacing $u := (-1) \uparrow n$ by

$u :=$ **if** $n = n \div 2 \times 2$ **then** 1 **else** -1

(iii) eliminating the separate evaluation of $n!$ by including the evaluation of $a^n \cdot (n!)^{-1}$ in the main loop. This gives a simpler argument for *sqrt* in the final assignment statement.

The revised algorithm then reads

```
real procedure PCpolynomial (x, n, a);
  value x, n, a; real x, a; integer n;
begin integer j; real u, s, c;
  s := u := if n = n ÷ 2 × 2 then 1 else −1;
  c := 1;
  for j := 0 step 1 until n − 1 do
  begin u := −u × (n−j) × (x−j)/(a × (j+1));
    s := s + u;
    c := c × a/(j+1)
  end;
  PCpolynomial := sqrt(c) × s
end PCpolynomial
```

This version gave the same results as the original but was appreciably faster.

CERTIFICATION OF ALGORITHM 236 [S17] BESSEL FUNCTIONS OF THE FIRST KIND [Walter Gautschi, *Comm. ACM 7* (Aug. 1964), 479]

Walter Gautschi (Recd. 24 Aug. 1964 and 2 Nov. 1964)
Purdue University, Lafayette, Ind.

All procedures were tested on the CDC 1604-A computer, using the Oak Ridge Algol compiler.

1. The procedure *Japlusn* was submitted to the following tests:

(a) Values of $J_n(2)$ and $J_{n+1/2}(10)$ were produced for $n = 0(1)10$, calling for an accuracy of $d = 6$ significant digits. The values obtained for $J_n(2)$ agreed with those of Table 9.4 in [1] to 10 significant digits (with occasional discrepancies of one unit in the tenth figure). The results for $J_{n+1/2}(10)$ were compared against those of $J_{n+1/2}(10) = 2.523132521 \times j_n(10)$ obtained from Table 10.5 in [1]. The maximum discrepancy was found to be five units in the tenth figure, occurring for $n = 3$.

(b) To observe the performance of the procedure near a zero of a Bessel function, we generated $J_n(x)$, $n = 0(1)10$, for $x = 2.40482556$—the 8D value of the first zero $j_{0,1}$ of J_0—calling for $d = 10$ significant digits. The results are shown in the table below.

n	$J_n(j_{0,1})$	n	$J_n(j_{0,1})$
0	$-1.1936252775_{10}-9$	6	$3.4048184902_{10}-3$
1	$5.1914749680_{10}-1$	7	$6.0068836955_{10}-4$
2	$4.3175480738_{10}-1$	8	$9.2165787385_{10}-5$
3	$1.9899990578_{10}-1$	9	$1.2517271082_{10}-5$
4	$6.4746666371_{10}-2$	10	$1.5253656182_{10}-6$
5	$1.6389243276_{10}-2$		

The entry for $n = 1$ agrees to 9 figures with that of $-J_0'(j_{0,1})$ given in Table 9.5 of reference [1].

(c) We drove the procedure to calculate $J_{x+\nu-1}(x)$ to 6 significant digits, for $x = 4(4)20$, $\nu = 0(.1)1.9$. The results agreed with those tabulated in [2].

2. The procedure *Iaplusn* was called to generate test values to 6 significant figures of $I_n(20)$, $I_{n+1/2}(10)$, $I_{n+1/4}(.1)$, for $n = 0(1)10$. The first two sets of values were compared with those in [3] and in Table 10.10 of [1], respectively, and found to be in error by at most 5 units in the tenth figure. The value for $I_{1/4}(.1)$ agreed to 10 figures with that given in [5].

3. Further checks were made on the procedures *Japlusn*, *Iaplusn*, as well as the procedures *Jaminusn*, *Iaminusn*, by having them "verify" the relation

$$f_{2a+2}(2x) = f_{a+1}^2(x) + 2\sum_{n=0}^{\infty} f_{a-n}(x) f_{a+n+2}(x)$$

for $x = 1$, $a = .2(.2).8$, where $f_\nu(x)$ stands for either $J_\nu(x)$ or $I_\nu(x)$ (cf. [4], p. 100, formula (21)). That is, we printed the relative errors incurred when the infinite series is truncated after the $(N+1)$-st term, $N = 0(5)20$. Selected results (rounded to four digits) are shown in the table below.

a \ N	0	5	10	15	20
.2	$1.165_{10}-2$	$2.519_{10}-4$	$-3.568_{10}-5$	$1.043_{10}-5$	$-4.234_{10}-6$
.8	$-7.945_{10}-2$	$4.968_{10}-5$	$-3.459_{10}-6$	$6.517_{10}-7$	$-1.923_{10}-7$
.4	$-8.091_{10}-2$	$1.245_{10}-4$	$-1.456_{10}-5$	$3.714_{10}-6$	$-1.361_{10}-6$
.6	$-1.023_{10}-1$	$7.590_{10}-5$	$-7.041_{10}-6$	$1.553_{10}-6$	$-5.115_{10}-7$

The first two lines refer to $f = J$, the last two lines to $f = I$. The driver program follows.

```
begin integer n; real a, sumJ, sumI, sJ, sI, errorJ, errorI;
  array J1, I1[0:3], J2, I2[0:22], J3, I3[0:20];
  for a := .2 step .2 until .9 do
  begin
    if 2 × a < 1 then
      begin
        Japlusn (2.0, 2 × a, 2, 6, J1); Iaplusn (2.0, 2 × a, 2, 6, I1);
        sumJ := J1[2]; sumI := I1[2]
      end
    else
      begin
        Japlusn (2.0, 2 × a−1, 3, 6, J1);
        Iaplusn (2.0, 2 × a−1, 3, 6, I1);
        sumJ := J1[3]; sumI := I1[3]
      end;
    Japlusn (1.0, a, 22, 6, J2); Jaminusn (1.0, a, 20, 6, J3);
    Iaplusn (1.0, a, 22, 6, I2); Iaminusn (1.0, a, 20, 6, I3);
    sJ := sI := 0;
    for n := 0 step 1 until 20 do
    begin
      sJ := sJ + J3[n] × J2[n+2]; sI := sI + I3[n] × I2[n+2];
      if entier (n/5) = n/5 then
```

```
        begin
          errorJ  :=  (J2[1]↑2 + 2 × sJ−sumJ)/sumJ;
          errorI  :=  (I2[1]↑2 + 2 × sI−sumI)/sumI;
          outstring (1, 'a='); outreal (1, a);
          outstring (1, 'N='); outinteger (1, n);
          outstring (1, 'errorJ='); outreal (1, errorJ);
          outstring (1, 'errorI='); outreal (1, errorI)
        end
    end
  end;
  go to skip;
alarm: outstring (1, 'parameters not in range');
skip: end
```

4. The procedure *Complex Japlusn* underwent the following tests:

(a) Values of $J_n(re^{i\phi})$ were produced for $n = 0, 1$, $\phi = (r-2) \times 30°$, $r = 1(1)6$, calling for an accuracy of 6 significant digits. Comparison with [6] showed agreement to 9–10 significant figures.

(b) We asked the procedure to "verify" the identity (cf. [4], p. 99, formula (2))

$$(z/2)^a J_0(z) = \sum_{n=0}^{\infty} \frac{\Gamma(1-a)\Gamma(a+n)}{(n!)^2\Gamma(1-a-n)}(a+2n)J_{a+2n}(z),$$

by printing the moduli of the relative errors incurred when truncating the infinite series at $n = 0(1)5$. We let a and z run through values $a = .2(.2).8$, $z = 2\exp(i\phi)$, $\phi = -150°\ (30°)\ 150°$, respectively. Selected results (rounded to three figures) are displayed in the table below.

$\phi°$	a \ n	0	1	2	3	4	5
−120	.2	$1.17_{10}-1$	$5.51_{10}-3$	$1.31_{10}-4$	$1.85_{10}-6$	$1.72_{10}-8$	$2.02_{10}-10$
−30	.4	$3.16_{10}-1$	$2.02_{10}-2$	$5.61_{10}-4$	$8.70_{10}-6$	$8.64_{10}-8$	$5.27_{10}-10$
60	.6	$2.60_{10}-1$	$1.65_{10}-2$	$4.67_{10}-4$	$7.41_{10}-6$	$7.51_{10}-8$	$3.93_{10}-10$
150	.8	$4.95_{10}-1$	$4.00_{10}-2$	$1.29_{10}-3$	$2.23_{10}-5$	$2.41_{10}-7$	$1.75_{10}-9$

The same pattern persists throughout the range of the variables. The driver program follows.

```
begin integer m, n; real a, phi, c, s, x, y, sum1, sum2,
  q, s1, s2, p, error; array u, v[0:10];
  for a := .2 step .2 until .9 do
  for m := −5 step 1 until 5 do
  begin
    phi  :=  .52359877560 × m;
    c := cos(a×phi); s := sin(a×phi);
    x := 2 × cos(phi); y := 2 × sin(phi);
    Complex Japlusn (x, y, 0, 0, 6, u, v);
    sum1 := c × u[0] − s × v[0]; sum2 := c × v[0] + s × u[0];
    Complex Japlusn (x, y, a, 10, 6, u, v);
    q := gamma (1+a);
    s1 := q × u[0]; s2 := q × v[0]; p := q/a;
    n := 0;
  L: error := sqrt (((sum1−s1)↑2 + (sum2−s2)↑2)/(sum1↑2
    + sum2↑2));
    outstring (1, 'a='); outreal (1, a);
    outstring (1, 'phi='); outinteger (1, 30×m);
    outstring (1, 'n='); outinteger (1, n);
    outstring (1, 'error='); outreal (1, error);
    n := n + 1;
    if n ≦ 5 then
    begin
      p := −p × ((n+a−1)/n)↑2; q := (a+2×n) × p;
      s1 := s1 + q × u[2×n]; s2 := s2 + q × v[2×n];
      go to L
    end
  end;
  go to skip;
alarm: outstring (1, 'parameters not in range');
skip: end
```


References:

1. Abramowitz, M., and Stegun, I. A. (Eds.) *Handbook of Mathematical Functions.* NBS Appl. Math. Ser. 55, U.S. Govt. Printing Off., Washington, D.C., 1964.
2. Airey, J. R. Bessel functions of nearly equal order and argument. *Philos. Mag.* (7) *19* (1935), 230–235.
3. BAAS. *Bessel functions, part II, Functions of positive integer order.* Mathematical Tables, vol. X, Cambridge U. Press, London, 1952.
4. Erdélyi, A. (Ed.) *Higher Transcendental Functions, vol. II.* McGraw-Hill, New York, 1953.
5. National Bureau of Standards. *Tables of Bessel functions of fractional order, vol. II.* Columbia U. Press, New York, 1949.
6. ——. *Table of the Bessel Functions $J_0(z)$ and $J_1(z)$ for Complex Arguments.* Columbia U. Press, New York, 1943.

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.