ALGORITHM 236
BESSEL FUNCTIONS OF THE FIRST KIND [S17]

WALTER GAUTSCHI (Recd. 10 Aug. 1963 and 10 Apr. 1964)
Oak Ridge National Laboratory, Oak Ridge, Tenn.*

* Now at Purdue University, Lafayette, Ind.

```
real procedure t(y); value y; real y;
comment This is an auxiliary procedure which evaluates the
  inverse function t = t(y) of y = t ln t  (t≧1) to an accuracy of
  about 1%. For the interval 0 ≦ y ≦ 10 a fifth degree approxi-
  mating polynomial was obtained by truncating a series expan-
  sion in Chebyshev polynomials. For y > 10 the approximation
  t(y) ≐ (y/ln(y/α))(1+(lnα−lnln(y/α))/(1+ln(y/α)))^−1 where
  ln α = .775† is used;
begin real p, z;
  if y ≦ 10 then
    begin
      p := .000057941 × y − .00176148;  p := y × p + .0208645;
      p := y × p − .129013;  p := y × p + .85777;
      t := y × p + 1.0125
    end
  else
    begin
      z := ln (y) − .775;  p := (.775−ln (z))/(1+z);
      p := 1/(1+p);  t := y × p/z
    end
end t;

procedure Japlusn (x, a, nmax, d, J); value x, a, nmax, d;
  integer nmax, d; real x, a; array J;
comment This procedure evaluates to d significant digits the
  Bessel functions J_{a+n}(x) for fixed a, x and for n = 0, 1, ··· , nmax.
  The results are stored in the array J. It is assumed
  that 0 ≦ a < 1, x > 0, and nmax ≧ 0. If any of these variables is
  not in the range specified, control is transferred to a nonlocal
  label called alarm. The procedure makes use of the real procedure
  t. In addition, it calls for a nonlocal real procedure gamma which
  evaluates Γ(z) for 1 ≦ z ≦ 2. (See [2].) The method of computa-
  tion is a variant of the backward recurrence algorithm of J. C.
  P. Miller. (See [1].) The purported accuracy is obtained by a
  judicious selection of the initial value ν of the recursion index,
  together with at least one repetition of the recursion with ν re-
  placed by ν + 5. Near a zero of one of the Bessel functions
  generated, the accuracy of that particular Bessel function may
  deteriorate to less than d significant digits. The algorithm is
  most efficient when x is small or moderately large;
begin integer n, nu, m, limit; real epsilon, sum, d1, r, s, L,
  lambda; array Japprox, Rr[0:nmax];
  if a < 0 ∨ a ≧ 1 ∨ x ≦ 0 ∨ nmax < 0 then go to alarm;
  epsilon := .5 × 10↑(−d);
  for n := 0 step 1 until nmax do Japprox[n] := 0;
  sum := (x/2)↑a/gamma (1+a);
  d1 := 2.3026 × d + 1.3863;
  if nmax > 0 then r := nmax × t(.5×d1/nmax) else r := 0;
  s := 1.3591 × x × t(.73576×d1/x);
  nu := 1 + entier (if r≦s then s else r);
L0: m := 0;  L := 1;  limit := entier (nu/2);
L1: m := m + 1;
  L := L × (m+a)/(m+1);
  if m < limit then go to L1;
  n := 2 × m;  r := s := 0;
L2: r := 1/(2×(a+n)/x−r);
  comment Conceivably, but very unlikely, division by an
    exact zero, or overflow, may take place here. The user may
    wish to test the divisor for zero, and, if necessary, enlarge it
    slightly to avoid overflow, before this statement is carried out.
    As such a test depends on the particular machine used, it was
    not included here;
  if entier (n/2) ≠ n/2 then lambda := 0 else
    begin
      L := L × (n+2)/(n+2×a);
      lambda := L × (n+a)
    end;
  s := r × (lambda+s);  if n ≦ nmax then Rr[n−1] := r;
  n := n − 1;  if n ≧ 1 then go to L2;
  J[0] := sum/(1+s);
  for n := 0 step 1 until nmax − 1 do J[n+1] := Rr[n] × J[n];
  for n := 0 step 1 until nmax do
    if abs((J[n] − Japprox[n])/J[n]) > epsilon then
    begin
      for m := 0 step 1 until nmax do Japprox[m] := J[m];
      nu := nu + 5;  go to L0
    end
end Japlusn;

procedure Iaplusn(x, a, nmax, d, I); value x, a, nmax, d;
  integer nmax, d; real x, a; array I;
comment This procedure evaluates to d significant digits the
  modified Bessel functions I_{a+n}(x) for fixed a, x, with 0 ≦ a < 1,
  x > 0, and for n = 0, 1, ··· , nmax. The results are stored in the
  array I. For the setup of the procedure, and the method of com-
  putation used, see the comment in Japlusn;
begin integer n, nu, m; real epsilon, sum, d1, r, s, L, lambda;
  array Iapprox, Rr[0:nmax];
  if a < 0 ∨ a ≧ 1 ∨ x ≦ 0 ∨ nmax < 0 then go to alarm;
  epsilon := .5 × 10↑(−d);
  for n := 0 step 1 until nmax do Iapprox[n] := 0;
  sum := exp(x) × (x/2)↑a/gamma(1+a);
  d1 := 2.3026 × d + 1.3863;
  if nmax > 0 then r := nmax × t(.5×d1/nmax) else r := 0;
  s := if x < d1 then 1.3591 × x × t(.73576×(d1−x)/x) else
    1.3591 × x;
  nu := 1 + entier (if r≦s then s else r);
L0: n := 0;  L := 1;
L1: n := n + 1;
  L := L × (n+2×a)/(n+1);
  if n < nu then go to L1;
  r := s := 0;
```

† In an earlier version of this procedure the author used α = 1. The value ln α = .775 was found empirically by H. C. Thacher, Jr. to yield somewhat better approximations.

```
L2: r := 1/(2×(a+n)/x+r);
  L := L × (n+1)/(n+2×a);
  lambda := 2 × (n+a) × L;
  s := r × (lambda+s); if n ≦ nmax then Rr[n−1] := r;
  n := n − 1; if n ≧ 1 then go to L2;
  I[0] := sum/(1−s);
  for n := 0 step 1 until nmax − 1 do I[n+1] := Rr[n] × I[n];
  for n := 0 step 1 until nmax do
    if abs((I[n]−Iapprox[n])/I[n]) > epsilon then
      begin
        for m := 0 step 1 until nmax do Iapprox[m] := I[m];
        nu := nu + 5; go to L0
      end
end Iaplusn;

procedure Jaminusn(x, a, nmax, d, J); value x, a, nmax, d;
  integer nmax, d; real x, a; array J;
comment This procedure evaluates to d significant digits the
  Bessel functions J_{a−n}(x) for fixed a, x, with 0 < a < 1, x > 0,
  and for n = 0, 1, ··· , nmax. The results are stored in the array
  J. The procedure makes use of the real procedure t, and the
  procedure Japlusn. In addition, it calls for a nonlocal real pro-
  cedure gamma which evaluates Γ(z) for 1 ≦ z ≦ 2. (See [2].) The
  accuracy may deteriorate to less than d significant digits if a is
  close to 0 or 1;
begin integer n; array J1[0:1];
  if a = 0 then go to alarm;
  Japlusn(x, a, 1, d, J1);
  J[0] := J1[0];
  J[1] := 2 × a × J[0]/x − J1[1];
  for n := 1 step 1 until nmax − 1 do
    J[n+1] := 2 × (a−n) × J[n]/x − J[n−1]
end Jaminusn;

procedure Iaminusn(x, a, nmax, d, I); value x, a, nmax, d;
  integer nmax, d; real x, a; array I;
comment This procedure evaluates to d significant digits the
  modified Bessel functions I_{a−n}(x) for fixed a, x, with 0 < a < 1,
  x > 0, and for n = 0, 1, ··· , nmax. The results are stored in the
  array I. The procedure makes use of the real procedure t, and
  the procedure Icplusn. In addition, it calls for a nonlocal real
  procedure gamma which evaluates Γ(z) for 1 ≦ z ≦ 2. (See [2].)
  The accuracy may deteriorate to less than d significant digits if
  a is close to 0 or 1;
begin integer n; array I1[0:1];
  if a = 0 then go to alarm;
  Iaplusn(x, a, 1, d, I1);
  I[0] := I1[0];
  I[1] := 2 × a × I[0]/x + I1[1];
  for n := 1 step 1 until nmax − 1 do
    I[n+1] := 2 × (a−n) × I[n]/x + I[n−1]
end Iaminusn;

procedure Complex Japlusn(x, y, a, nmax, d, u, v); value x, y, a,
  nmax, d;
  integer nmax, d; real x, y, a; array u, v;
comment This procedure evaluates to d significant digits the
  Bessel functions J_{a+n}(z) = u_n + iv_n for fixed real a, fixed complex
  z = x + iy, and for n = 0, 1, ··· , nmax. The real parts u_0,
  u_1, ··· , u_nmax of the results are stored in the array u, the imagi-
  nary parts v_0, v_1, ··· , v_nmax in the array v. It is assumed that
  0 ≦ a < 1, nmax ≧ 0, and that z is not on the negative real axis
  x ≦ 0, y = 0. Otherwise, control is transferred to the nonlocal
  label alarm upon entry of the procedure. The procedure makes
  use of the real procedure t. In addition, it calls for a nonlocal
  real procedure gamma which evaluates Γ(z) for 1 ≦ z ≦ 2. (See
  [2].) The method of computation is a complex extension of the
  method used in the procedure Japlusn. The algorithm is most
  efficient when | z | is small or moderately large;
begin integer n, nu, m; real epsilon, y1, r02, r0, phi, c, c1, c2,
  sum1, sum2, d1, r, s, lambda1, lambda2, L, r1, r2, s1, s2; array
  uapprox, vapprox, Rr1, Rr2[0:nmax];
  if a < 0 ∨ a ≧ 1 ∨ (x≦0∧y=0) ∨ nmax < 0 then go to alarm;
  epsilon := .5 × 10 ↑ (−d);
  for n := 0 step 1 until nmax do uapprox[n] := vapprox[n] := 0;
  y1 := abs(y); r02 := x ↑ 2 + y ↑ 2; r0 := sqrt(r02);
  phi := if x = 0 then 1.5707963268 else if x > 0 then arctan(y1/x)
    else 3.1415926536 + arctan(y1/x);
comment The two constants π/2 and π in the preceding state-
  ment are to be supplied with the full accuracy desired in the
  final results;
  c := exp(y1) × (r0/2)↑a/gamma (1+a);
  sum1 := c × cos(a×phi−x); sum2 := c × sin(a×phi−x);
  d1 := 2.3026 × d + 1.3863;
  if nmax > 0 then r := nmax × t(.5×d1/nmax) else r := 0;
  s := if y1 < d1 then 1.3591 × r0 × t(.73576×(d1−y1)/r0) else
    1.3591 × r0;
  nu := 1 + entier (if r≦s then s else r);
L0: n := 0; L := 1; c1 := 1; c2 := 0;
L1: n := n + 1;
  L := L × (n+2×a)/(n+1);
  c := −c1; c1 := c2; c2 := c;
  if n < nu then go to L1;
  r1 := r2 := s1 := s2 := 0;
L2: c := (2×(a+n)−x×r1+y1×r2) ↑ 2 + (x×r2+y1×r1) ↑ 2;
  r1 := (2×(a+n)×x−r02×r1)/c;
  r2 := (2×(a+n)×y1+r02×r2)/c;
  L := L × (n+1)/(n+2×a); c := 2 × (n+a) × L;
  lambda1 := c × c1; lambda2 := c × c2;
  c := c1; c1 := −c2; c2 := c;
  s := r1 × (lambda1+s1) − r2 × (lambda2+s2);
  s2 := r1 × (lambda2+s2) + r2 × (lambda1+s1);
  s1 := s;
  if n ≦ nmax then begin Rr1[n−1] := r1; Rr2[n−1] := r2 end;
  n := n − 1;
  if n ≧ 1 then go to L2;
  c := (1+s1) ↑ 2 + s2 ↑ 2;
  u[0] := (sum1×(1+s1)+sum2×s2)/c;
  v[0] := (sum2×(1+s1)−sum1×s2)/c;
  for n := 0 step 1 until nmax − 1 do
    begin
      u[n+1] := Rr1[n] × u[n] − Rr2[n] × v[n];
      v[n+1] := Rr1[n] × v[n] + Rr2[n] × u[n]
    end;
  if y < 0 then for n := 0 step 1 until nmax do v[n] := − v[n];
  for n := 0 step 1 until nmax do
    if sqrt(((u[n]−uapprox[n]) ↑ 2+(v[n]−vapprox[n]) ↑ 2)
      /(u[n] ↑ 2+v[n] ↑ 2)) > epsilon
    then
    begin
      for m := 0 step 1 until nmax do
        begin uapprox[m] := u[m]; vapprox[m] := v[m] end;
      nu := nu + 5; go to L0
    end
end Complex Japlusn
```

REFERENCES

1. GAUTSCHI, W. Recursive computation of special functions. U. Mich. Engineering Summer Conferences, Numerical Analysis, 1963.
2. ——. Algorithm 221—Gamma function. *Comm. ACM 7* (Mar. 1964), 143.

ALGORITHM 237
GREATEST COMMON DIVISOR [A1]

J. E. L. Peck (Recd. 16 Dec. 1963)
University of Alberta, Calgary, Alberta, Canada

```
integer procedure Euclidean (a) dimension : (n) linear coefficients : (x); value a; integer array a, x; integer n;
comment This procedure finds the greatest common divisor of the n nonnegative elements of the vector a, and produces values for x_i in the expression (a_1 , a_2 , ··· , a_n) = a_1x_1 + a_2x_2 + ··· + a_nx_n;
begin integer array M[1:n, 1:n];
  integer i, j, min, max, imin, imax, q, t;
  comment We set up M as an identity matrix;
INITIALISE:
  for i := 1 step 1 until n do
    for j := 1 step 1 until n do M[i, j] := 0;
  for i := 1 step 1 until n do M[i, i] := 1; max := 0;
  comment We search for the least nonzero integer in the array a. Note that this step need not be repeated at every iteration (see statement labelled DIVIDES);
MINIMUM:
  for i := 1 step 1 until n do
    begin t := a[i];
    if t ≠ 0 ∧ (max=0∨t<max) then
      begin max := t; imax := i end
    end of minimum search. If the use of the identifier max is confusing, observe the two statements following the label MAXIMUM, where the confusion is resolved;
    if max = 0 then go to ERROR; comment ERROR is a global label;
MAXIMUM: imin := imax; min := max;
  comment We search for the greatest element of a;
  max := a[1]; imax := 1;
  for i := 2 step 1 until n do if a[i] > max then
    begin max := a[i]; imax := i end of maximum search;
  if max ≠ min then
REDUCTION:
    begin comment Note that the identity a_i = ∑_{j=1}^{n} m_ij a_j holds at each stage of the reduction;
    q := max ÷ min; a[imax] := max := max − q × min;
    for j := 1 step 1 until n do
      M[imax, j] := M[imax, j] − q × M[imin, j];
DIVIDES: go to if max = 0 then MINIMUM else MAXIMUM
    end of the reduction. Note that if max ≠ 0 then max now contains the new nonzero minimum.
If max = min then we are ready with the results;
for j := 1 step 1 until n do x[j] := M[imin, j];
Euclidean := min
end of procedure Euclidean
```

REFERENCE

1. Blankinship, W. A. A new version of the Euclidean algorithm. *Amer. Math. Mon. 70* (1963), 742–745.

ALGORITHM 238
CONJUGATE GRADIENT METHOD [F4]

C. M. Reeves (Recd. 18 Nov. 1963)
Electronic Computing Lab., Univ. of Leeds, England

```
procedure conjugate gradients (x, r, n, matmult);
value n; real array x, r; integer n; procedure matmult;
comment The method of conjugate gradients [cf: Beckman, F. S. Mathematical Methods for Digital Computers. Ch. 4, Ralston, A., and Wilf, H. S., (Eds.), Wiley 1960.] is applied to solve the equations Ax = b where A is a general nonsingular matrix of order n, and x and b are vectors. At entry x contains an initial approximation to the solution, and r contains b, the vector of constants. Both x and r have bounds [1:n]. Up to n+1 iterations are carried out and at exit the solution is in x and the corresponding residuals r = b − Ax are in r.
  The procedure matmult has the following heading, with semicolons which must now be omitted:
  procedure matmult (transpose, dat, res)
  Boolean transpose real array dat, res
  comment The datum vector dat is premultiplied by the matrix B and the result formed in res where, denoting the transpose of A by At,

                B = if transpose then At else A

  The body of matmult will depend upon whether A is stored on magnetic tape, and whether all or only its nonzero elements are stored. The products should be accumulated in double precision, if possible.;
begin integer iterations; real alpha, beta, At r sq;
  real array p, temp [1:n];
  real procedure dot (u, v);
  real array u, v;
  comment dot is the scalar product of the vectors u and v;
  begin integer i; real sum; sum := 0;
    for i := 1 step 1 until n do sum := sum + u[i] × v[i];
    dot := sum
  end of dot;
  procedure combine (f) plus: (c) times: (g) to form: (h);
  value c;
  real c; real array f, g, h;
  comment f + cg is formed in h;
  begin integer i;
    for i := 1 step 1 until n do h[i] := f[i] + c × g[i]
  end of combine;
Start:
  for iterations := 0 step 1 until n do
  begin if iterations = 0
    then begin matmult (false, x) in : (temp);
        combine (r, −1, temp) in : (r);
        matmult (true, r) in : (p);
        At r sq := dot (p, p);
      end of forming r = b − Ax, p = At r, and At r sq
    else begin matmult (true, r) giving At r in : (temp);
        beta := dot (temp, temp)/At r sq;
        combine (temp, beta, p) in : (p);
        At r sq := beta × At r sq
      end;
    if At r sq = 0 then go to finish;
    matmult (false, p) giving Ap in : (temp);
    alpha := dot (temp, temp);
    if alpha = 0 then go to finish;
    alpha := dot (r, temp)/alpha;
    combine (x, alpha, p) in : (x);
    combine (r, −alpha, temp) in : (r)
  end of iterative loop;
finish :
end of conjugate gradients;
```

ALGORITHM 239
FREE FIELD READ [I5]

W. M. McKeeman (Recd. 12 Dec. 63 and 1 May 1964)
Computation Center, Stanford University, Stanford, Calif.

```
procedure inreal (channel, destination); value channel;
  integer channel; real destination;
begin comment Each invocation of inreal will read one ⟨number⟩ [Revised Report ··· ALGOL 60, section 2.5.1] from the input
```

```
medium designated by the parameter channel and convert it
into the internal machine representation appropriate for real
numbers. Successive data values within the data string are
separated by the blank character ␣. Integer values from the
input medium are converted into values of type real. A nonlocal
procedure error is invoked whenever a non-⟨number⟩ is en-
countered in the input string. The action of error is left un-
defined;
real sig, fp, d;
integer esig, ep, ip, ch;
integer procedure CHAR;
begin comment The value of CHAR is the integer repre-
  senting the next character from the input string. insymbol
  is defined in the "Report on Input-Output Procedure for ALGOL
  60," ALGOL Bull. No. 16 (May 1964), 9-13; Comm. ACM, to ap-
  pear. Characters occurring in the second parameter of in-
  symbol are mapped onto the integers corresponding to their
  position, left-to-right, within the string. Other basic symbols
  map onto the integer 0.
    The present procedure inreal differs from the inreal of
  the referenced Report on Input-Output Procedures for
  ALGOL 60 in the following ways:
    (a) The report does not specify what values may be pre-
  sented in its inreal, only that whatever is presented will be
  assigned to the second parameter of inreal. I demand that a
  ⟨number⟩ be presented.
    (b) No separator of values on the foreign medium is speci-
  fied. I demand an ALGOL string blank.;
  real c;
  insymbol (channel, '0123456789.-+10␣'    , c);
  if c ≦ 0 then error; comment an illegal character;
  CHAR := c - 1
end CHAR;
integer procedure unsigned integer;
begin comment ⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨unsigned
  integer⟩⟨digit⟩;
  integer u;
  u := 0;
K: u := 10 × u + ch;
  ch := CHAR;
  if ch < 10 then go to K;
  unsigned integer := u
end unsigned integer;
sig := 1.0; ep := 0; fp := 0;
L: ch := CHAR;
  if ch = 14 then go to L; comment suppress initial blanks;
  comment ⟨number⟩ ::= ⟨unsigned number⟩ | +⟨unsigned
  number⟩ | -⟨unsigned number⟩;
  if ch = 12 then ch := CHAR
  else if ch = 11 then
  begin comment 12 = "+" and 11 = "-";
    sig := -1.0;
    ch := CHAR
  end;
  comment ⟨unsigned number⟩ ::= ⟨decimal number⟩ | ⟨ex-
  ponent part⟩ | ⟨decimal number⟩⟨exponent part⟩;
  if ch ≦ 10 then
  begin comment ⟨decimal number⟩ ::= ⟨unsigned integer⟩ |
  ⟨decimal fraction⟩ | ⟨unsigned integer⟩⟨decimal fraction⟩;
    if ch < 10 then ip := unsigned integer else ip := 0;
    if ch = 10 then
    begin comment ⟨decimal fraction⟩ ::= .⟨unsigned integer⟩;
      ch := CHAR;
      if ch ≧ 10 then error; comment a digit must follow the
        ".";
      fp := 0; d := 0.1;
M:    fp := fp + ch × d;
      d := d × 0.1;
      comment a table of reciprocal powers of ten is preferable
        to the statement d := d × 0.1;
      ch := CHAR;
      if ch < 10 then go to M
    end
  end else if ch = 13 then ip := 1 else error;
  if ch = 13 then
  begin comment ⟨exponent part⟩ ::= 10⟨integer⟩;
    ch := CHAR; esig := 1;
    comment ⟨integer⟩ ::= ⟨unsigned integer⟩ | +⟨unsigned
      integer⟩ | -⟨unsigned integer⟩;
    if ch = 12 then ch := CHAR
    else if ch = 11 then
    begin comment negative exponent;
      esig := -1;
      ch := CHAR
    end;
    if ch < 10 then ep := unsigned integer × esig else error
  end;
  if ch ≠ 14 then error; comment the required "␣" separator;
  destination := sig × (ip+fp) × 10.0 ↑ ep
end inreal
```

REMARK ON ALGORITHM 162 [J6]
XYMOVE PLOTTING [F. G. Stockton, *Comm. ACM 6* (Apr. 1963), 161; *6* (Aug. 1963), 450]

D. K. CAVIN (Recd. 10 Feb. 1964)
Oak Ridge National Laboratory, Oak Ridge, Tenn.

The following modifications were made to Algorithm 162 to decrease the average execution time. The last nine lines of Algorithm 162 are replaced by the following:

```
          move := code(I-1); I := code(I);
repeat:   A := D + E; B := T + E + A;
          if B ≧ 0 then begin E := A; F := F - 2; plot(I) end
            else begin E := E + T; F := F - 1; plot(move) end;
          if F > 0 then go to repeat;
return:
end
```

It is obvious that on any movement containing more than two elemental pen movements the use of the code procedure in the loop is redundant, since no more than two of the eight permitted pen movements are necessary for the approximation of any line. Therefore moving the call of the code procedure outside of the basic loop reduces the execution time whenever the X, Y movement requires more than two elemental pen movements. The procedures were coded in CODAP, the assembly language for the CDC 1604-A, and this modified version was approximately 40 percent faster in the loop than the original version. The timing comparisons used numbers in the range −2000 to 2000 with heavy emphasis on the subrange −150 to 150. The typographical error noted in the certification (*Comm. ACM*, August 1963) was corrected in both codes.

[A referee verifies that Algorithm 162 does indeed run, as changed.—G.E.F.]

CERTIFICATION OF ALGORITHM 209 [S15]
GAUSS [D. Ibbetson, *Comm. ACM 6*, Oct. 1963, 616]

M. C. PIKE
Statistical Research Unit of the Medical Research Council, University College Hospital Medical School, London, England

This procedure was tested on an Elliott 803 computer using the

Please turn to page 485

ADDITIONAL BIBLIOGRAPHY

Date	Title (or Subject)	Prepared By
3/9/64	Proposed American Standard for Bit Sequencing of ASCII in Serial-by-Bit Data Transmission	X3.3/4
3/10/64	Letter Ballot on Publication of Proposed Standard for Bit Order Sequencing of ASCII	C. A. Phillips
3/19/64	Statement of Position (UNIVAC)	E. H. Clamons
3/20/64	Analysis of the Report of the Select Committee on ASCII Bit Order of Data Transmission	L. W. Claussen F. C. White
3/26/64	ASCII	S. N. Alexander
4/2/64	Letter to E. H. Clamons re. Clamon's memorandum to X3, March 19, 1964	V. G. Grey
4/9/64	Reply to ASA letter of April 2, 1964	E. H. Clamons
(undated)	Letter to C. A. Phillips re. Letter Ballot on Publication of Proposed Standard for Bit Order Sequencing of ASCII	G. L. Bowlby
4/10/64	Letter to C. A. Phillips re. Letter Ballot on Publication of Proposed Standard for Bit Order Sequencing of ASCII	R. W. Ferguson
4/10/64	Letter to C. A. Phillips re. Letter Ballot on Publication of Proposed Standard for Bit Order Sequencing of ASCII	L. Wolff
(undated)	Explanation for Negative Vote on Acceptability for Publication of Proposed Standard for Bit Sequencing of ASCII	L. W. Claussen
4/14/64	Results of Ballot on Acceptability for Publication of Proposed Standard on Bit Sequencing of ASCII and Approval of Publication Period	C A. Phillips

Editor's Note

Publication of the following proposed American Standards, developed by a Subcommittee of ASA Sectional Committee X3, has been authorized by the American Standards Association for the purpose of obtaining comment, criticism and general public reaction, with the understanding that such proposed American Standards have not been finally accepted by ASA as standards and, therefore, are subject to change, modification or withdrawal in whole or in part. Comments should be addressed to the Secretary, Business Equipment Manufacturers Association, 235 East 42 Street, New York 17, N.Y.—E.L.

PROPOSED AMERICAN STANDARDS

Interchangeable Perforated Tape Variable Block Formats for Positioning and Straight Cut (RS-273) and Contouring and Contouring/Positioning (RS-274) Numerically Controlled Machine Tools

These standards are intended to serve as guides in the coordination of system design, to minimize the variety of program manuscripts required and the number of word and block format systems used, to promote uniformity of programming techniques, and to foster interchangeability of input tapes between numerically controlled machine tools of the same classification by type, process function, size and accuracy. It is intended that simple numerically controlled machine tools be programmed using a simple format which is systematically extensible for more complex machine tools.

These standards apply wherever a variable block format is used on perforated tape to control positioning and straight cut and contouring or contouring/positioning numerically controlled machine tools. These formats will usually be used with tape read row-by-row.

Perforated tape with variable block format as described shall be usable interchangeably among numerically controlled machine tools which conform to the same format classification as described in the standards. (Note: The degree of interchangeability will depend upon the conformity of the machines with respect to function, capacity, range, horsepower, geometric relationship of axes, preparatory, miscellaneous, and tooling functions, and use of absolute or incremental dimensions.)

The interchangeable tape described in these standards is a combination of word address and tab sequential format, and includes both tab and address characters.

These standards cover interchangeable perforated tape variable block format only, and are not intended to specify machine tool design. In certain cases provisions must be included by the control system builder and machine tool builder, in order to gain full use of interchangeable perforated tape.

[Full texts of these two standards are available from Electronic Industries Association, Engineering Department, 11 West 42 Street, New York 36, N. Y.: RS-273, $1.10; RS-274, $1.30.]

ALGORITHMS—Cont'd from page 482

standard Elliott ALGOL compiler. The expression

$$2 \times Gauss\ (x) - 1$$

was evaluated for $x = 0(.01)6$ and the answers checked with those given in *Tables of Probability Functions, vol. II,* U.S. National Bureau of Standards, Washington, D.C., 1942, where they are given to 15 decimal places. There was a maximum error of 1 in the 8th decimal place.

REMARK ON REMARKS ON ALGORITHM 48 [B3] LOGARITHM OF A COMPLEX NUMBER [John R. Herndon, *Comm. ACM 4* (Apr. 1961), 179; *5* (Jun. 62), 347; *5* (Jul. 62), 391]

DAVID S. COLLENS (Recd. 24 Jan. 1964 and 1 Jun. 1964)
Computer Laboratory, The University, Liverpool 3, England

This procedure was designed to compute $\log_e(a+bi)$, namely $c+di$, and although some very necessary precautions about its use have already been stated, some points seem to have escaped notice. In particular, A. P. Relph [*Comm. ACM*, June 1962, 347] remarked that if $a = 0$, then c becomes '−infinity', but this is only the case if $b = 0$ also. Margaret L. Johnson and Ward Sangren [*Comm. ACM*, July 1962, 391] conceded that $a = b = 0$ was a special case, but wrongly gave zero as the result. The only reasonable way of dealing with this case is to exit to some nonlocal label and to let the user decide whether to terminate his program or to assign particular values to c and d. The obvious values to use here are, for c, a negative number, larger than the largest which would be given by the procedure, and possibly zero for d. (In an implementation where 2^{-129} is the smallest representable nonzero number, the largest negative value of c possible is −89.416.) Finally, in the Johnson-Sangren version of the procedure, the last conditional statement should read

if $a = 0 \wedge b < 0$ **then begin** $c := ln(abs(b))$;

$d := -1.570963$; **go to** *RETURN* **end**;

the omission of the minus sign in the original being probably typographical in origin.