Algorithms

J. H. WEGSTEIN, Editor

ALGORITHM 160
COMBINATORIAL OF M THINGS TAKEN N AT A TIME

M. L. WOLFSON AND H. V. WRIGHT
United States Steel Corp., Monroeville, Penn.

```
integer procedure combination (m, n);
value n; integer m, n;
comment calculates the number of combinations of m things
  taken n at a time. If n is less than half of m, then the program
  calculates the combinations of m things taken m − n at a time
  which is the exact equivalent of m things taken n at a time;
begin integer p, r, i;
  p := m − n;
  if n < p then begin p := n; n := m − p end;
  if p = 0 then begin r := 1; go to exit end;
  r := n + 1;
  for i := 2 step 1 until p do r := (r × (n+i))/i;
exit: combination := r
end combination
```

ALGORITHM 161
COMBINATORIAL OF M THINGS TAKEN ONE AT A TIME, TWO AT A TIME, UP TO N AT A TIME

H. V. WRIGHT AND M. L. WOLFSON
United States Steel Corp., Monroeville, Penn.

```
procedure combination vector (m, n, v);
integer m, n; integer array v;
comment calculates all combinations of m things taken from 1
  to n at a time. The result is a vector, v, within which the first
  element is the combination of m things taken 1 at a time, the
  second element is the combinations of m things taken 2 at a time,
  the third element taken 3 at a time, ···, and the nth element
  taken n at a time.
begin integer i;
  v[1] := m;
  for i := 2 step 1 until n do
    v[i] := (v[i−1] × (m−i+1))/i;
end combination vector
```

ALGORITHM 162
XYMOVE PLOTTING

FRED G. STOCKTON
Shell Development Co., Emeryville, Calif.

```
procedure xymove (XZ, YZ, XN, YN); value XZ, YZ, XN, YN;
integer XZ, YZ, XN, YN;
comment xymove computes the code string required to move the
  pen of a digital incremental X,Y-plotter from an initial point
  (XZ, YZ) to a terminal point (XN, YN) by the "best" approxi-
  mation to the straight line between the points. The permitted
  elemental pen movement is to an adjacent point in a plane
  Cartesian point lattice, diagonal moves permitted. The eight
  permitted pen movements are coded

      1 = +Y,  2 = +X+Y,  3 = +X,  4 = +X−Y,
      5 = −Y,  6 = −X−Y,  7 = −X,  8 = −X+Y.

  The approximation is "best" in the sense that each point tra-
  versed is at least as near the true line as the other candidate
  point for the same move.
  xymove does not use multiplication or division.;
begin integer A, B, D, E, F, T, I, move;
comment code (J) is a procedure which returns a value of code
  according to the following table:
```

J	1	2	3	4	5	6	7	8
code	1	2	3	2	3	4	5	4
J	9	10	11	12	13	14	15	16
code	5	6	7	6	7	8	1	8

```
  plot (move) is a procedure which sends move to the plotter as a
  plotter command.;
    if XZ = XN ∧ YZ = YN then go to return;
    A := XN − XZ; B := YN − YZ; D := A + B; T :=
    B − A; I := 0;
    if B ≧ 0 then I := 2;
    if D ≧ then I := I + 2;
    if T ≧ 0 then I := I + 2;
    if A ≧ 0 then I := 8 − I else I := I + 10;
    A := abs(A); B := abs(B); F := A + B; D := B − A;
    if D ≧ 0 then begin T := A; D := −D end else T := B;
    E := 0;
  repeat: A := D + E; B := T + E + A;
          if B ≧ 0 then begin E := A; move := code(I);
          F := F − 2 end
          else begin E := E + T; F := F − 1;
          move := code(I−1) end;
          plot(move);
          if F > 0 then go to repeat;
  return:
end
```

ALGORITHM 163
MODIFIED HANKEL FUNCTION

HENRY E. FETTIS
Aeronautical Research Laboratories, Wright-Patterson Air Force Base, Ohio

```
procedure EXPK (P, X, E); real P, X, E;
comment this procedure calculates the modified Hankel Function
  e^x K_p(x) to within a given accuracy E from the integral
  representation:
```

$$e^{x}K_{p}(x) = \int_{0}^{\infty} e^{x(1-\cosh t)} \cosh(pt)\,dt;$$

```
begin real F, G, H, R, S, T, U, Y, Z, ZP;
  R := 0.0;
H := 1.0;
    iteration: begin
        G := R;
        T := .5 × H;
        S := 0;
        Z := exp(T);
        U := Z × Z;
          integration: begin
            Y := X × (1−.5×(Z+1/Z));
            if P = 0 then ZP := 1
            else ZP := Z ↑ P;
            F := .5 × exp(Y) × (ZP+1/ZP);
            S := S + F;
            Z := Z × U;
            end;
            if F ≧ E then go to integration
            else R := H × S;
            H := .5 × H;
          end;
        if abs (R−G) ≧ E then go to iteration
        else EXPK: = R
  end  EXPK
```

ALGORITHM 164
ORTHOGONAL POLYNOMIAL LEAST SQUARES SURFACE FIT

R. E. Clark, R. N. Kubik, L. P. Phillips
The Babcock & Wilcox Co., Atomic Energy Div., Lynchburg, Va.

```
procedure surfacefit (x, u, y, w, z, nmax, mmax, imax, jmax)
  result: (beta, phi, zcomp, minsqd, minsqdcomp, sumdifcomp,
  maxdifcomp);
real array x, u, y, w, z, phi, beta, zcomp;
integer nmax, mmax, imax, jmax;
real minsqd, minsqdcomp, sumdifcomp, maxdifcomp;
comment this is a transliteration of an operating program written in Burroughs Algol for the B-220. It fits, in the least squares
  sense, a polynomial function of two independent variables to
  values of a dependent variable specified at points on a rectangular grid in the plane of the independent variables. The use of
  orthogonal polynomials leads to a particularly simple system of
  linear equations rather than the ill-conditioned system which
  arises from the usual normal equations. It also provides a measure of the improvements resulting from each new term included
  which further leads, in this algorithm, to an automatic selection
  of a "best" degree polynomial function as determined by
  Gauss' criterion. The initial normalization of the variables results in significant reduction of round off errors in many cases.
  This algorithm is developed more fully in BAW-182. For a very
  similar approach to this and related problems see Cadwell,
  J. H., "Least Squares Surface Fitting Program", The Computer
  J. 3 (1961), 266 and Cadwell, J. H., and Williams, D. E., "Some
  Orthogonal Methods of Curve and Surface Fitting," The Computer J. 4 (1961), 260. A further reference is Gauss, C. F., "Theoria Combinationis Observationum Erroribus Minimis Obnoxial,"
  Gauss Werke 4 (Gottingen 1873), 3-93. x[i] and y[j] are the independent variables, z[i, j] is the dependent variable. u[i] and
  w[j] represent the weights corresponding to x[i] and y[j], respectively. nmax is one more than the maximum degree of x to be
  considered. mmax is one more than the maximum degree of y to
  be considered. imax is the number of x's, and jmax is the number
  of y's. beta[n, m] is a measure of the improvement resulting
  from the inclusion of the x^n y^m th term. phi[n, m] is the polynomial coefficient for the x^n y^m th term. Note the degree of the
  resulting polynomial may be less than the maximum degree
  specified as a result of the application of Gauss' criterion.
  zcomp is the computed dependent variable.
```

$$minsqd = \left(\frac{\sum_{i,j} u[i] \cdot w[j] \cdot z[i,j]^2 - \sum_{n,m} beta[n,m]}{imax \cdot jmax} \right)^{1/2}$$

$$minsqdcomp = \left(\frac{\sum_{i,j} u[i] \cdot w[j] (z[i,j] - zcomp[i,j])^2}{imax \cdot jmax} \right)^{1/2}$$

$$sumdifcomp = \frac{\sum_{i,j} | z[i,j] - zcomp[i,j] |}{imax \cdot jmax}$$

$$maxdifcomp = max \, | z(i,j) - zcomp((i,j) |$$

```
  minsqd and minsqdcomp are equal if computation is exact. In
  practice they will not be equal due to the imprecise nature of
  calculation. A wide discrepancy indicates excessive errors in
  calculation;
begin
real array a, b, denpa[1:nmax], c, d, denqa[1:mmax],
  alpha[1:nmax, 1:mmax], p[1:nmax, 1:imax], q[1:mmax, 1:jmax],
  pc[1:nmax, 1:nmax], qc[1:mmax, 1:mmax];
integer n, m, i, j, s, t, r;
real sumx, sumy, sumz, meanx, meany, meanz, numa, dena, denb,
  numc, denc, dend, alph, sumzsq, gausscrit, trialgausscrit, betasum,
  rescomp, poly;
comment normalization of variables;
sumx := sumy := sumz := 0.0;
for i := 1 step 1 until imax do
  sumx := sumx + x[i];
meanx := sumx/imax;
for i := 1 step 1 until imax do
  x[i] := x[i] − meanx;
for j := 1 step 1 until jmax do
  sumy := sumy + y[j];
meany := sumy/jmax;
for j := 1 step 1 until jmax do
  y[j] := y[j] − meany;
for i := 1 step 1 until imax do begin
  for j := 1 step 1 until jmax do
    sumz := sumz + z[i,j] end;
meanz := sumz/(imax × jmax);
for i := 1 step 1 until imax do begin
  for j := 1 step 1 until jmax do
    z[i, j] := z[i, j] − meanz end;
comment evaluate orthogonal polynomials;
numa := dena := 0.0;
for i := 1 step 1 until imax do begin
  p[1, i] := 1.0;
  numa := numa + u[i] × x[i];
  dena := dena + u[i] end;
a[2] := numa/dena;
for i := 1 step 1 until imax do
  p[2, i] := x[i] − a[2];
for n := 3 step 1 until nmax do begin
  numa := dena := denb := 0.0;
  for i := 1 step 1 until imax do begin
    numa := numa + u[i] × x[i] × p[n−1] ↑ 2;
    dena := dena + u[i] × p[n−1, i] ↑ 2;
    denb := denb + u[i] × p[n−2, i] ↑ 2 end;
    a[n] := numa/dena; b[n] := dena/denb;
    for i := 1 step 1 until imax do
      p[n, i] := (x[i]−a[n]) × p[n−1, i] − b[n] × p[n−2, i] end;
  numc := denc := 0.0;
  for j := 1 step 1 until jmax do begin
    q[1, j] := 1.0;
    numc := numc + w[j] × y[j];
    denc := denc + w[j] end;
```

```
  c[2] := numc/denc;
  for j := 1 step 1 until jmax do
    q[2, j] := y[j] - c[2];
  for m := 3 step 1 until mmax do begin
    numc := denc := dend := 0.0;
    for j := 1 step 1 until jmax do begin
      numc := numc + w[j] × y[j] × q[m-1, j] ↑ 2;
      denc := denc + w[j] × q[m-1, j] ↑ 2;
      dend := dend + w[j] × q[m-2, j] ↑ 2 end;
    c[m] := numc/denc; d[m] := denc/dend;
    for j := 1 step 1 until jmax do
      q[m, j] := (y[j]-c[m]) × q[m-1, j] - d[m] × q[m-2, j] end;
comment evaluate contribution of each orthogonal polynomial
  to the minimization of the residuals;
for n := 1 step 1 until nmax do begin
  denpa[n] := 0.0;
    for i := 1 step 1 until imax do
    denpa[n] := denpa[n] + u[i] × p[n, i] ↑ 2 end;
  for m := 1 step 1 until mmax do begin
    denqa[m] := 0.0;
    for j := 1 step 1 until jmax do
      denqa[m] := denqa[m] + w[j] × q[m, j] ↑ 2 end;
  for n := 1 step 1 until nmax do begin
    for m := 1 step 1 until mmax do begin
      alph := 0.0;
      for i := 1 step 1 until imax do begin
        for j := 1 step 1 until jmax do
          alph := alph + u[i] × w[j] × z[i, j] × p[n, i] × q[m, j]
        end;
    alpha[n, m] := alph/(denpa[n]×denqa[m]);
    beta[n, m] := alpha[n, m] × alph; end end;
comment application of Gauss' criterion to determine the de-
  gree polynomial which yields the closest fit to the given data.
  Gauss' criterion is, strictly speaking, applicable only to cases
  where the weights u[i] and w[j] are unity;
sumzsq := 0.0;
for i := 1 step 1 until imax do begin
  for j := 1 step 1 until jmax do
    sumzsq := sumzsq + u[i] × w[j] × z[i, j] ↑ 2 end;
s := t := 1;
for n := 1 step 1 until nmax do begin
  betasum := 0.0;
  for m := 1 step 1 until mmax do begin
    for r := 1 step 1 until n do
      betasum := betasum + beta[r, m];
    if betasum > sumzsq then trialgausscrit := 0.0
    else
    trialgausscrit := (sumzsq-betasum)/(imax×jmax-n×m);
    if n = 1 ∧ m = 1 then gausscrit := trialgausscrit;
    if gausscrit = trialgausscrit then begin
      if n × m < s × t then begin
        s := n;
        t := m end end;
    if gausscrit > trialgausscrit then begin
      gausscrit := trialgausscrit;
      s := n;
      t := m end end end;
nmax := s;
mmax := t;
minsqd := (gausscrit×(imax×jmax-nmax×mmax)/(imax×jmax))
  ↑ ½;
comment evaluation of orthogonal polynomial coefficients;
for n := 1 step 1 until nmax do begin
  pc[n, m] := 1.0;
  for s := 1 step 1 until n - 1 do begin
    pc[n, s] := -a[n] × pc[n-1, s];
    if s ≠ 1 then pc[n, s] := pc[n, s] + pc[n-1, s-1];
    if s ≠ n - 1 then pc[n, s] := pc[n, s] - b[n] × pc[n-2, s]
    end end;
for m := 1 step 1 until mmax do begin
  qc[m, m] := 1.0;
  for t := 1 step 1 until m - 1 do begin
    qc[m, t] := -c[m] × qc[m-1, t];
    if t ≠ then qc[m, t] := qc[m, t] + qc[m-1, t-1];
    if t ≠ m - 1 then qc[m, t] := qc[m, t] - d[m] × qc[m-2, t]
    end end;
comment evaluation of approximating polynomial coefficients;
for s := 1 step 1 until nmax do begin
  for t := 1 step 1 until mmax do begin
    phi[s, t] := 0.0;
    for n := s step 1 until nmax do begin
      for m := t step 1 until mmax do
        phi[s, t] := phi[s, t] + alpha[n, m] × pc[n, s] × qc[m, t]
        end end end;
comment evaluation of dependent variables using the approxi-
  mating polynomial;
minsqdcomp := sumdifcomp := maxidifcomp := 0.0;
for i := 1 step 1 until imax do begin
  for j := 1 step 1 until jmax do begin
    zcomp[i, j] := 0.0;
    for s := nmax step - 1 until 1 do begin
      poly := phi[s, mmax];
      for t := mmax - 1 step 1 until 1 do
        poly := poly × y[j] + phi[s, t];
      zcomp[i, j] := zcomp[i, j] × x[i] + poly end;
    rescomp := z[i, j] - zcomp[i, j];
    zcomp[i, j] := zcomp[i, j] + meanz;
    minsqdcomp := minsqdcomp + u[i] × w[j] × rescomp ↑ 2;
    sumdifcomp := sumdifcomp + abs(rescomp);
    if abs (rescomp) > maxdifcomp then
    maxdifcomp := abs(rescomp) end end;
minsqdcomp := (minsqdcomp/(imax × jmax)) ↑ ½;
sumdifcomp := sumdifcomp/(imax × jmax);
end surfacefit
```

ALGORITHM 165
COMPLETE ELLIPTIC INTEGRALS

HENRY C. THACHER, JR.*
Reactor Eng. Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U.S. Atomic Energy Commission.

```
procedure KANDE(ml, K, E, tol, alarm);
value ml, tol;
real ml, K, E, tol;
label alarm;
comment this procedure computes the complete elliptic inte-
  grals K(ml) = ∫₀^{π/2} (1 - (1 - ml) sin²ν)^{-1/2} dν and E(ml) =
  ∫₀^{π/2} (1 - (1 - ml) sin²ν)^{1/2} dν by the arithmetic-geometric-
  mean process. The accuracy is limited only by the accuracy of
  the arithmetic.
```

Except for the provision of tests for pathological values of the parameter, the calculation of K is only a slight modification of the second procedure of Algorithm No. 149 (*Comm. ACM.* **5** (Dec. 1962), 605). These integrals may also be approximated to limited (6D) accuracy by Algorithms 55 and 56 (Comm. ACM. **4** (Apr. 1961), 180). Unless the square-root is exceptionally fast, the latter algorithms are probably more efficient for 6D-accuracy.

The complementary parameter, $m1$, is chosen as the independent variable, rather than the parameter, m, the modulus, k or the modular angle α, because of the possibility of serious loss of significance in generating $m1$ from the other possible independent variables when $m1$ is small and $dK/dm1$ is very large. These variables are related by $m1 = 1 - m = 1 - k^2 = \cos^2\alpha$.

```
    The formal parameter, tol, determines the relative accuracy
  of the result. To prevent entering a nonterminating loop, tol
  should not be less than twice the relative error in the square
  root routine. If m1 ≦ 0 or if m1 > 1, the procedure exits to
  alarm. K(0) = ∞ while E(0) = 1.00000000.
    The body of this procedure has been tested using the Dart-
  mouth SCALP processor for the LGP-30. With tol = 5₁₀ − 7,
  results agreed with tabulated values to within 3 in the seventh
  significant digit;
begin real a, b, c, sum, temp;
  integer fact;
  if m 1 > 1 ∨ m1 ≦ 0 then go to alarm;
  a := fact := 1;
  b := sqrt(m1);
  temp := 1 − m1;
  sum := 0;
iter: sum := sum + temp;
  c := (a − b)/2;
  fact := fact + fact;
  temp := (a + b)/2;
  b := sqrt (a × b);
  a := temp;
  temp := fact × c × c;
  if abs(c) ≧ tol × a ∨ temp > tol × sum then go to iter;
  sum := sum + temp;
  K := 3.141592654/(a + b);
  comment pi must be given to the full accuracy desired;
  E := K × (1 − sum/2)
end
```

ALGORITHM 166
MONTECARLO

R. D. RODMAN
Burroughs Corp., Pasadena, Calif.

```
procedure montecarlo (n, a, row, tol, mxm, inv, test, count);
  value n, row, tol, mxm; integer n, row, mxm, count;
  real tol; real array a, inv, test;
  comment this procedure will compute a single row of the
    inverse of a given matrix using a monte carlo technique.
    n is the size of the matrix, array a is the matrix, row indicates
    which inverse row is to be computed, tol is a tolerance factor
    and thus a criterion for terminating the process, mxm is 1000
    times the maximum number of random walks to be taken,
    after which the process is terminated, array inv contains the
    inverse row, array test contains the innerproduct of inv with
    the rowth column of a, count is the number of random walks
    executed upon termination. real procedure RANDOM must
    be declared in the blockhead of procedure MONTE CARLO
    and generates a single random number between 0 and 1. If
    a is the matrix to be inverted, the absolute value of the largest
    eigenvalue of the matrix I − a (I is the unit matrix) must be
    less than one to assure convergence. This procedure is easily
    adapted to finding a single unknown from a set of simultaneous
    linear equations;
  begin integer i, j, k, nwk, lastwalk, walk; real res, p, g;
  real array sum[1:n], v[1:n, 1:n];
start: p := (n−1)/n × n;
       for i := 1 step 1 until n do for j := 1 step 1 until n do
       v[i,j] := if i ≠ j then −a[i,j]/p else (1−a[i,j])/p;
       nwk := 1000;
       count := res := 0;
       for k := 1 step 1 until n do test [k] := sum [k] := 0;
start1: lastwalk := row; g := 1;
start2: walk := (RANDOM/p) + 1;
       if walk > n then go to stop;
       g := v[lastwalk,walk] × g; lastwalk := walk;
       go to start2;
stop:  count := count + 1; sum[lastwalk] := sum[lastwalk] + g;
       if count < nwk then go to start1;
       for k := 1 step 1 until n do inv[k] := n × sum[k]/count;
       for i := 1 step 1 until n do for k := 1 step 1 until n do
       test[i] := inv[k] × a[k, i] + test[i];
       for i := 1 step 1 until row−1, row+1 step 1 until n do
       res := abs(test[i]) + res; res := abs(test[row]−1) + res;
       if res < tol then go to exit;
       if count ≧ 1000 × mxm then go to exit;
       nwk := nwk + 1000; res := 0;
       for k := 1 step 1 until n do test [k] := 0;
       go to start1;
exit:  end of monte carlo inversion procedure
```

ALGORITHM 167
CALCULATION OF CONFLUENT DIVIDED DIFFERENCES

W. KAHAN AND I. FARKAS
Institute of Computer Science, University of Toronto, Canada

```
real procedure DVDFC(n, X, V, B, W); integer n;
  real array X, V, B, W;
comment DVDFC ca.culates the forward divided difference
  Δf(X₁ , X₂ , ··· Xₙ). n is an integer which takes the values
  n = 1, 2, 3, ··· in turn. X is a real array of dimension at least
  n in which X[i] = Xᵢ for i = 1, 2, ··· , n. The values Xᵢ need
  not be distinct nor in any special order, but once the array X
  is chosen it will fix the interpretation of the arrays B and V.
  If X[1], X[2], ··· , X[n] are in monotonic order, then the effect
  of roundoff upon any nth divided difference is no more than
  would be caused by perturbing each f(X[i]) by n units at most
  in its last significant place. But if the X's are not in mono-
  tonic order, the error can be catastrophic if some of the divided
  differences are relatively large. V is a real array of dimension
  at least n containing the values of the function f(X) and per-
  haps its derivatives at the point Xᵢ. V[i] = f^m(Xᵢ)/m! and
  m = mᵢ for i = 1, 2, 3, ··· , n. mᵢ is the number of times that
  the value of Xᵢ has previously appeared in the array X. B is
  a real array of dimension at least n containing backward divided
  differences. Before a reference to DVDFC is executed one should
  have B[i] = Δf(Xᵢ , Xᵢ₊₁ , ··· , Xₙ₋₁) for i = 1, 2, ··· , n−1.
  After that reference to DVDFC is executed one will find B[i] =
  Δf(Xᵢ , Xᵢ₊₁ , ··· , Xₙ₋₁ , Xₙ) for i = 1, 2, ··· , n−1, n. When
  n = 1 the initial state of B is irrelevant. W is a real array of
  dimension (2 + m̄) at least, where m̄ is the maximum value of
  mᵢ for i = 1, 2, ··· , n. W is used for work space;
    begin real DENOM; integer i, j, NK, NIN;
    if n = 1 then go to L1;
    NK := 1;
    for i := 1 step 1 until n do
    begin
    if X[i] = X[n] then begin NK := NK + 1;
      W[NK] := V[i] end
    end i;
    for i := n step −1 until 2 do
    begin W[1] := B[i − 1]; B[i] := W[2];
    NIN := if n − i + 2 < NK then n − i + 2 else NK;
    for j := NIN step −1 until 2 do
    begin
    DENOM := X[n] − X[i + j − 3];
    if DENOM ≠ 0 then go to L2;
```

```
        W[j] := W[j + 1];
        if NK − j − 1 ≠ 0 then go Cont;
        NK := NK − 1;
        go to Cont;
  L2:   W[j] := (W[j] − W[j − 1])/DENOM;
Cont:   end j
        end i;
        B[1] := W[2];
        go to L3;
  L1:   B[1] := V[1];
  L3:   DVDFC := B[1]
        end DVDFC
```

The following program segment is an example of how *DVDFC* can be used to construct a table of forward or backward differences.

```
for n := 1 step 1 until N do
begin
X[n] := ··· ; V[n] := ··· ; F[n] := DVDFC(n, X, V, B, W)
end;
```

The array F can be used in $FNEWT(z, N, X, F, R, D, E)$ or the array B in $BNEWT(z, N, X, B, P, D, E)$. See algorithms "Newton interpolation with forward (backward) divided differences." *DVDFC* has been written as a FORTRAN II function and is available from I.C.S., University of Toronto;

ALGORITHM 168
NEWTON INTERPOLATION WITH BACKWARD DIVIDED DIFFERENCES

W. KAHAN AND I. FARKAS
Institute of Computer Science, University of Toronto, Canada

procedure *BNEWT*(z, N, X, B, P, D, E); **value** z, N; **real** z, P, D, E; **integer** N; **real array** X, B;

comment X is a real array of dimension at least N in which $X[i] = X_i$ for $i = 1, 2, 3, \cdots, N$. The values X_i need not be distinct nor in any special order, but once the array X is chosen it will fix the interpretation of the array B. B is a real array of dimension at least N and contains the backward divided differences $B[i] = \Delta f(X_i, X_{i+1}, \cdots, X_N)$ $i = 1, 2, \cdots, N$. If two or more of the values X_i are equal then some of the B's must be confluent divided differences, see algorithm: "Calculation of confluent divided differences." P is the value of the following polynomial in z of degree $N-1$ at most, $B(N) + (z-X_N)\cdot\{B(N-1) + (z-X_{N-1})\{B(N-2) + \cdots + (z-X_2)B(1)\}\cdots\}$. This polynomial is an interpolation polynomial which would, but for rounding errors, match values of the function $f(x)$ and any of its derivatives that *DVDFC* might have been given. D is the value of the derivative of P. E is the maximum error in P caused by roundoff during the execution of *BNEWT*. The error estimate is based upon the assumption that the result of each floating point arithmetic operation is truncated to 27 significant binary digits as is the case in FORTRAN programs on the 7090. *BNEWT* has been written as a FORTRAN II subroutine and is available from I.C.S., University of Toronto;

```
begin real z1; integer i;
P := D := E := 0;
for i := 1 step 1 until N do
begin
z1 := z − X[i];
D := P + z1 × D;
P := B[i] + z1 × P;
E := abs(P) + E × abs(z1)
end;
E := (1.5×E − abs(P))×3₁₀ − 8
end BNEWT
```

ALGORITHM 169
NEWTON INTERPOLATION WITH FORWARD DIVIDED DIFFERENCES

W. KAHAN AND I. FARKAS
Institute of Computer Science, University of Toronto, Canada

procedure *FNEWT*(z, N, X, F, R, D, E); **value** z, N; **real** z, R, D, E; **integer** N; **real array** X, F;

comment X is a real array of dimension at least N in which $X[i] = X_i$ for $i = 1, 2, \cdots, N$. The values X_i need not be distinct nor in any special order, but once the array X is chosen it will fix the interpretation of the array F. F is a real array of dimension at least N and contains the forward divided differences $F[i] = \Delta f(X_1, X_2, \cdots, X_i)$ $i = 1, 2, \cdots, N$. If two or more of the values X_i are equal then some of the F's must be confluent divided differences, see algorithm: "Calculation of confluent divided differences." R is the value of the following polynomial in z of degree $N-1$ at most, $F(1) + (z-X_1)\cdot\{F(2) + (z-X_2)\{F(3) + \cdots + (z-X_{N-1})F(N)\}\cdots\}$. This polynomial is an interpolation polynomial which would, but for rounding errors, match values of the function $f(x)$ and any of its derivatives that *DVDFC* might have been given. D is the value of the derivative of R. E is the maximum error in R caused by roundoff during the execution of *FNEWT*. The error estimate is based upon the assumption that the result of each floating-point arithmetic operation is truncated to 27 significant binary digits as is the case in FORTRAN programs on the 7090. *FNEWT* has been written as a FORTRAN II subroutine and is available from I.C.S., University of Toronto;

```
begin real z1; integer i;
R := D := E := 0;
for i := N step −1 until 1 do
begin
z1 := z − X[i];
D := R + z1 × D;
R := F[i] + z1×R;
E := abs(R) + abs(z1)×E
end;
E := (1.5×E − abs(R))×3₁₀ − 8
end FNEWT
```

ALGORITHM 170
REDUCTION OF A MATRIX CONTAINING POLYNOMIAL ELEMENTS

PAUL E. HENNION
Giannini Controls Corp., Astromechanics Res. Div., Berwyn, Penn.

real procedure *POLYMATRIX* (A, $NCOL$, N, COE, $NP1$); **value** A, $NCOL$, N; **real array** A; **integer** $NCOL$, N;

comment this procedure will expand a general determinant, where each of the elements are polynomials in the Laplace complex variable. This program is useful for the investigation of dynamic stability problems when using the transfer function approach. The process is one of triangularization of a polynomial matrix with real coefficients whereupon multiplication of the diagonal elements the determinant polynomial is formed. The polynomial matrix as defined herein is a matrix whose elements are polynomials of the form $\sum_{i=0}^{N} a_i x^i$. When such a matrix is triangularized, all elements below the main diagonal are nulled. Then upon expanding, the nonvanishing terms are those formed by the product of these diagonal elements. Hence stability criteria may be checked by evaluating the roots of the characteristic equation thus formed using some suitable root extracting routine.

Consider the polynomial matrix with quadratic elements ($N = 2$). In this case the three-dimensional input matrix A is size $A[1{:}NCOL, 1{:}NCOL, 1{:}M]$, where $NCOL$ is the order of the matrix and $M = N \times NCOL + 1$. Here the first subscript of A refers to the row, the second to the column, and the third to the polynomial coefficient. Therefore, prior to entry the constant term of a general polynomial element is contained in $A[i, j, 1]$, the linear term is contained in $A[i, j, 2]$, and the quadratic term in $A[i, j, 3]$. Upon completion of the routine, the coefficients of the determinant polynomial are contained in $COE[1{:}M]$. The constant coefficient being in COE [1], the linear coefficient in COE [2], the quadratic coefficient in COE [3], etc. The variable $NP1$ will specify the number of coefficients of the determinant polynomial. In general $NP1 \neq M$ since some terms may vanish during the expansion.

If the polynomials comprising the matrix elements are not all of equal degree, set N prior to entry equal to the degree of the highest ordered polynomial;

```
begin real sa, sb; integer i, j, k, j1, j2, j3, j4, j5, j6, j7, j8, j9, j10,
  j11, NP1, M; array C1[1:M], C2[1:M], COE[1:M];
  integer array MAT [1:NCOL, 1:NCOL];
start: M := N×NCOL+1; for i := 1 step until NCOL do
  begin for j := 1 step 1 until NCOL do begin MAT [i,j] := 0;
  for k := 1 step 1 until M do begin
  if A[i, j, k]≠0 then MAT [i, j] := k; end end end; j1 := 1:
L0: j9 := 0; for i := j1 step 1 until NCOL do begin
  if MAT [i,j1]<0 then go to exit;
  else if MAT [i, j1] =0 then go to L1
  else j9 := j9+1; j3 := i;
L1: end; if (j9−1)<0 then go to exit
  else if (j9−1)>0 then go to L2
  else if (j3−j1)<0 then go to exit
  else if (j3−j1)=0 then go to L12
  else for j := j1 step 1 until NCOL do
  begin j2 := MAX(MAT [j3,j], MAT [j1,j]); j4 := MAT [j3,j];
  MAT [j3,j] := MAT [j1, j]; MAT [j1,j] := j4;
  for k := 1 step 1 until j2 do
  begin sa := A [j3, j,k]; A[j3,j,k] := A[j1, j, k];
  A[j1,j,k] := −sa; end end; go to L12;
L2: j3 := j1+1; for i := j3 step 1 until NCOL do begin
L3: if (MAT[i,j1])<0 then go to exit
  else if [MAT [i,j1])=0 then go to L11
  else if (MAT [j1,j1])<0 then go to exit
  else if (MAT[j1,j1])=0 then go to L4
  else if (MAT [i,j1] −MAT [j1,j1]) ≧ 0 then go to L5 else
L4: for j := j1 step 1 until NCOL do begin
  j2 := MAX(MAT [j1,j], MAT [i, j]); j4 := MAT [j1,j];
  MAT (j1,j] := MAT [i,j]; MAT [i,j] := j4;
  for k := 1 step 1 until j2 do begin sa := A[i,j,k];
  A[i,j,k] := A[j1,j,k]; A[j1,j,k] := −sa;
  end end; go to L3;
  comment Interchange row i with j1;
L5: j7 := MAT [i,j1]; j5 := MAT [j1,j1]; j6 := j7−j5;
  sb := A[i,j1,j7]/A[j1,j1,j5];
  if (abs(sb)−4)<0 then go to L6
  else if (j6)<0 then go to exit
  else if (j6)=0 then go to L4 else
L6: for j := j1 step 1 until NCOL do begin j5 := MAT [j1, j];
  for k := 1 step 1 until j5 do begin j7 := k+j6;
  if (j7−M)>0 then go to L10 else
L7: if (abs(A[i,j,j7] −sb×A[j1,j,k])−2₁₀−8)≦0 then go to L8
  else A[i,j,j7] := A[i,j,j7] −sb×A[j1,j,k];
  go to L9;
L8: A[i,j,j7] := 0;
L9: end end;
L10: for j := j1 step 1 until NCOL do begin
  j7 := MAX(MAT [i,j], MAT[j1,j]+j6); MAT [i,j] := 0;
  for k := 1 step 1 until M do begin if (A[i,j,k])≠0 then
  MAT [i,j] := k end end;
L11: end; go to L0;
L12: j1 := j1+1; if (j1−NCOL)<0 then go to L0 else
  for j := 1 step 1 until NCOL do begin
  j2 := MAT [j,j];
  for k := 1 step 1 until j2 do C1[k] := A[j,j,k];
L13: if (j−1)<0 then go to exit
  else if (j−1)=0 then go to L14
  else for k := 1 step 1 until NP1 do C2[k] := COE[k];
  for k := 1 step 1 until M do COE[k] := 0;
  if (j2)<0 then go to exit
  else if (j2)=0 then go to L15
  else for k := 1 step 1 until j2 do begin
  for j10 := 1 step 1 until NP1 do begin
  j11 := k+j10−1;
  COE[j11]:=COE[j11]+C1[k]×C2[j10];
  end end; NP1 := j11; go to L15;
L14: for k := 1 step 1 until j2 do COE [k] := C1 [k];
  NP1 := j2;
L15: end;
exit: end POLYMATRIX
```

CERTIFICATION OF ALGORITHM 55 COMPLETE ELLIPTIC INTEGRAL OF THE FIRST KIND [John R. Herndon, *Comm. ACM*, Apr. 1961] and CERTIFICATION OF ALGORITHM 149 COMPLETE ELLIPTIC INTEGRAL [J. N. Merner, *Comm. ACM*, Dec. 1962]

HENRY C. THACHER, JR.*
Reactor Eng. Div., Argonne National Laboratory, Argonne, Ill.

* Work supported by the U.S. Atomic Energy Commission.

The bodies of Algorithm 55 and of the second procedure of Algorithm 149 were tested on the LGP-30 computer using SCALP, the Dartmouth "LOAD-AND-GO" translator for a substantial subset of ALGOL 60. The floating-point arithmetic for this translator carries 7+ significant digits.

In addition to modifications required because of the limitations of the SCALP subset, the following need correction:

In Algorithm 55:

1. The constant 0.054555509 should be 0.054544409.
2. The function *log* should be *ln*.

In procedure ELIP 2 of Algorithm 149, the statement $a := c$ should be $a := C$.

The parameters of Algorithm 149 are related to the complete elliptic integral of the first kind by: $K = a \times ELIP(a, b)$ where the parameter $m = k^2 = 1 - b/a$.

The maximum approximation error in Algorithm 55 is given by Hastings as about $0.6_{10}-6$. In addition there is the possibility of serious cancellation error in forming the complementary parameter $t = 1 - k \times k$. For k near 1, errors as great as 4 significant digits were sustained. In these regions, the complementary parameter itself is a far more satisfactory parameter.

The accuracy obtainable with Algorithm 149 is limited only by the arithmetic accuracy and the amount of effort which it is desired to expend. Six-figure accuracy was obtained with 5 applications of the arithmetic-geometric mean for $a = 1000$, $b = 2$, and with one application for $a = 500$, $b = 500$.

Neither algorithm is satisfactory for $k = 1$. The behavior for Algorithm 55 will be governed by the error exit from the logarithm procedure. Under these circumstances, Algorithm 149 goes into an endless loop. Algorithm 149 may also go into an endless loop of the terminating constant ($_{10}-8$ in the published algorithm) is too small for the arithmetic being used. For the SCALP arithmetic it was found necessary to increase this tolerance to $5.0_{10}-7$. The

resulting values of the elliptic integrals were, however, accurate to within 2 in the 7th significant digit (6th decimal).

The relative efficiency of the two algorithms will depend strongly on the efficiency of the square-root and logarithm subroutines. With most systems, Algorithm 55 will provide sufficient accuracy, and will be more efficient. If a square-root operation or a highly efficient square-root subroutine is available, Algorithm 149 may well be the better method.

CERTIFICATION OF ALGORITHM 73 INCOMPLETE ELLIPTIC INTEGRALS [David K Jefferson, *Comm. ACM* Dec. 1961]

R. P. VAN DE RIET
Mathematical Centre, Amsterdam

The algorithm contained three misprints:
The 26th line of the procedure

$$E := (2 \times n-1)/(2 \times N);$$

should read

$$E := (2 \times n-1)/(2 \times n);$$

The 46th line of the procedure

$$\uparrow 2) \uparrow ((2 \times n-1)/(2 \times n));$$

should read

$$\uparrow 2) \uparrow ((2 \times n-1)/2)/(2 \times n);$$

The 49th line of the procedure

$$L[2] := L[4] + 1/(n \times 2 \times n-1));$$

should read

$$L[2] := L[1] + 1/(n \times (2 \times n-1));$$

The program was run on the X1 computer of the Mathematical Centre. For *phi* = 45°, *k* = *sin*(10°(10°)180°), *E* and *F* were calculated. The result contained 12 significant digits.

Comparison with a 12-decimal table of Legendre-Emde (1931) showed that the 12th digit was affected with an error, at most 4 units large. After about 10 minutes of calculation (i.e. more than 100 cycles) no results were obtained for *k* = *sin* 89°, *phi* = 1° and the calculation was discontinued.

REMARKS. As *phi* is unchanged during the calculation, we placed the statement *cos phi* := *cos* (*phi*) in the beginning of the program, to be certain that the cosine was not calculated 30 or more times. Moreover, in the expression for *T*[1] and *T*[2], *sqrt* (1-*sin phi* ↑ 2) was replaced by *cos phi*, so that loss of significant figures does not occur.

The expression $2 \times n$ was changed in a new variable, to obtain a more rapid program.

CERTIFICATION OF ALGORITHM 91 CHEBYSHEV CURVEFIT [A. Newhouse, *Comm. ACM*, May 1962]

ROBERT P. HALE
University of Adelaide, Adelaide, South Australia

The CHEBFIT algorithm was translated into FORTRAN and successfully run on an IBM 1620 when the following alterations were made:

(a) 2nd line after

comment Initialize;

should read

for *i* := 1 **step** 1 **until** *n*+1 **do** *IN*[*i*] := (*i*−1) × *k* + 1;

(b) 2nd and 3rd lines after

Poly: **comment** polynomial coefficients;

should read

begin *A*[*i*] := *AY*[*i*+1] + *AH*[*i*+1] × *H*; *BY*[*i*+1] := 0

(c) 3rd line after

ERROR: **comment** compute deviations;

should read

for *j* := 1 **step** 1 **until** *n* **do** *T*[*i*] := *T*[*i*] × X[*i*] + *A*[*n*−*j*];

(d) Immediately before statement *L*2 insert *i* := *n* + 2; (as **for** list may be exhausted and *i* no longer defined).

(e) 2nd line after statement *L*3 should read *IN*[1] := *imax*;

(f) 1st line after statement *L*4 should read *IN*[*i*−1] := *imax*;

CERTIFICATION OF ALGORITHM 133 RANDOM [Peter G. Behrenz, *Comm. ACM*, Nov. 1962]

JESSE H. POORE, JR.
Louisiana Polytechnic Institute, Ruston, La.

Algorithm 133 was transliterated into FORTRAN II for the IBM 1620 computer. A monitor program performed the test indicated in Algorithm 133 on the generated numbers.

Results of the test are shown in the following chart. The notation used is identical to that used in the algorithm.

X_0	$\frac{1}{N}\Sigma X_n$	$\frac{1}{N}\Sigma X^2{}_n$	
	.4986480931	.3280561242	N = 500
13543288579	.4840396640	.3141520616	N = 1000
	.4996829627	.3321160892	N = 5000
	.4971414796	.3297990588	N = 500
24376589411	.4997720126	.3326801987	N = 1000
	.4986380784	.3319949173	N = 5000
	.4962408228	.3339214302	N = 500
34359738367	.4974837457	.3335720239	N = 1000
	.4929612237	.3253421270	N = 5000
	.5313808305	.3691599122	N = 500
11324679915	.5167083685	.3498558251	N = 1000
	.5043814637	.3383429327	N = 5000

CERTIFICATION OF ALGORITHM 145 ADAPTIVE NUMERICAL INTEGRATION BY SIMPSON'S RULE [W. McKeeman, *Comm. ACM*, Dec. 1962]

WM. M. MCKEEMAN
Stanford University, Stanford, Calif.

Suggested changes in the code:

1. Replace all occurrences of *eps*/3.0 by *eps*/1.7.
2. Replace level ≧ 7 by level ≧ 20.
3. The second parameter *a* in the final call of Simpson was omitted; insert it.

With the above changes, a BALGOL translation of *Integral* has been tested successfully on a large number of functions. An example of its behavior is given below:

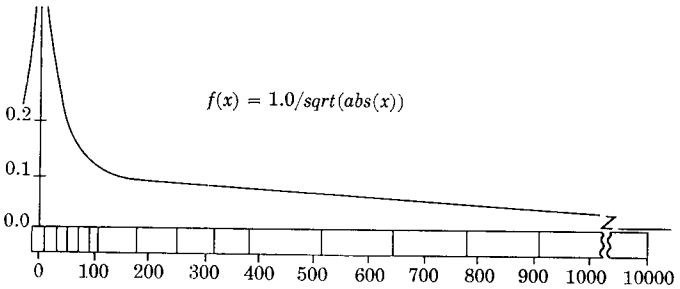
Machine: Burroughs 220, 8 decimal digit floating-point mantissa.
f(*x*) = 1.0/*sqrt*(*abs*(*x*)); which has a pole at the origin.
a = −9.0; *b* = 1000.0; correct answer = 206.0;

eps	computer answer	relative error
0.1	200.22251	0.028
0.01	206.00226	0.0000107
0.001	206.00092	0.0000045
0.0001	205.99985	0.0000007

If the recursion was allowed to go thirty levels deep we found:

0.0001 206.00005 0.0000002

The graph below shows the adaptive clustering of the points of evaluation around the pole of the function (taken from the first example above).



Each vertical line represents a point of evaluation for the function during the execution of the call:
integral(f, -9.0, 10000.0, 0.1);

CERTIFICATION OF ALGORITHM 147
PSIF [D. Amit, *Comm. ACM.*, Dec. 62]

HENRY C. THACHER, JR.*
Reactor Eng. Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

The following minor errors were noted in this algorithm:
a. (3) in the comment should read $\epsilon < 1/240\ x^8$.
b. The function *tan* is not a standard ALGOL function. It should be declared, or replaced by *sin* ( )/*cos*( ).
c. The block labelled *large* should be closed by inserting **end** immediately after 252.

The efficiency of the program would be improved by the following modifications:
a. Let the statement

if $x = 0$ **then begin** . . . **end**;

be the first statement of the procedure body.
b. Delete the condition $x \neq 0$ from the if clause,

if $x > -1 \wedge x \neq 0$ **then**. . .

c. Delete the declaration of *pei*, and the assignment of the value of 3.141592654 to *pei* in the statement
psi := *pei* × *sin*(*pei* × (*x*+0.5))/*cos*(*pei*×(*x*+0.5));
replace *pei* by the value 3.141592654.
d. Replace the block labelled large by:
large: **begin real** *y*; *x* := 1/*x*; *y* := *x* × *x*;
psi := *psi* − *ln*(*x*) + *x*/2 − ((*y*/252−0.008333333333) × *y* + 0.08333333333) × *y* **end**;

With these changes, the body of the procedure was translated and run on the LGP-30 computer using the Dartmouth SCALP processor. The program was used to tabulate *psif*(*x*) for $x = -1$(0.5)0(0.005)1.250. With $a = 3.0$ the results agreed with tabulated values to within 3 in the 6th decimal place. This is considered satisfactory, since one decimal place is lost in applying the recurrence. Running time, including output on the Flexowriter and computation of new values of the independent variable, averaged about 30 seconds per value.

It should be observed that *psif*(*x*) is $\Psi(x+1)$ as tabulated, for example, by Jahnke-Emde-Losch.

CERTIFICATION OF ALGORITHM 148
TERM OF MAGIC SQUARE [D. M. Collinson, *Comm. ACM*, Dec. 1962]

J. N. R. BARNECUT
University of Alberta, Calgary; Calgary, Alberta, Canada

MAGICTERM was translated into FORTRAN for the IBM 1620. The procedure was tested for terms of squares up to order 13. Correct results were obtained. For determination of complete squares operating time was not significantly different from Algorithm 118.

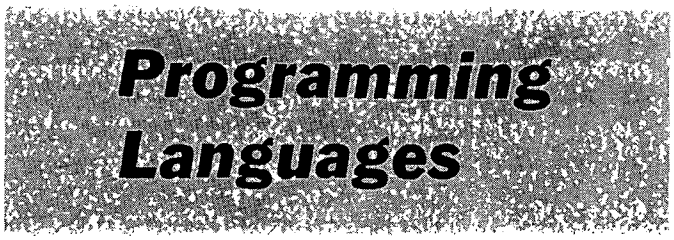
CERTIFICATION OF ALGORITHM 148
TERM OF MAGIC SQUARE [D. M. Collison, *Comm. ACM*, Dec. 62]

DMITRI THORO
San Jose State College, San Jose, Calif.

This algorithm was translated into FORTRAN and FORGO for the IBM 1620. No changes in the program were necessary. The elements of magic squares of odd orders up to 15 were generated satisfactorily.

A Suggested Method of Making Fuller Use of Strings in ALGOL 60

MIRIAM G. SHOFFNER AND PETER J. BROWN
University of North Carolina, Chapel Hill, N. C.



T. E. CHEATHAM, Jr., Editor

Notation. Throughout this paper string quotes (' ') are used only as basic ALGOL symbols, whereas double quotes (" ") have no significance other than as punctuation marks in the English language.

It seems to us that while ALGOL 60 cannot be made into a good symbol manipulation language without many additions two simple extensions would make a considerable difference, making ALGOL an adequate language for some symbol manipulation problems and also improving it in other ways. For a suggested way for incorporating full symbol manipulation features into ALGOL, see "A String Language for Symbol Manipulation Based on ALGOL 60" by J. H. Wegstein and W. W. Youden [*Comm. ACM*, Jan. 1962]. We propose two extensions to the use of strings which, we think, could be added into an existing translator with virtually no effort. As ALGOL stands, strings can be used only as actual parameters of procedures, presumably output procedures; nested strings are allowed but it would need some ingenuity to find a use for them. In our discussion we take "string" to mean a non-nested string. Our proposed additions (henceforward referred to here as "the additions") are:

(i) Strings can be assigned as values of variables.

(ii) Strings can be used as operands of the relational operators: "=" and "≠".

Examples:

color := *'black'*;

if *color* = *'black'* **then** ··· ;

A description of the exact syntactical changes necessary in the language is given at the end of this paper. The equivalent of the additions is already built into such algebraic languages as IT and GAT.

Variables having strings as value would be declared as being of type "**string**", and string arrays and string procedures (i.e. function designators of type string) would be allowed. It would be natural to have input procedures to assign strings as values of variables and output procedures to print string variables.

It might be a practical necessity for fixed-word-length machines to restrict the length of strings to one word-length. Restrictions somehow do not seem to be in the spirit of the ALGOL 60 report (to the exasperation of translator writers) but, of course, there are implicit restrictions on the size of integers and real numbers; so a restriction on the length of strings is not too unreasonable.

Note that the additions do not provide for a way to decompose strings within the language. However, "packing" and "unpacking" routines could be added to a system which would respectively break a string down and store it symbol by symbol in a string array, and perform the reverse operation. A large class of symbol manipulation problems can be performed efficiently by splitting input strings into a number of strings consisting of one symbol each and storing these in an array.

The following advantages would be gained by incorporating the additions.

(i) Simple string manipulation programs could be written in ALGOL. Programs have been written in GAT, which is a subset of ALGOL as extended, to formally differentiate, prove theorems, transpose music, and perform other such functions. ALGOL could be a very powerful language for writing such nonnumeric programs.

(ii) It would be possible to output variable alphabetic information, for instance information input as data. This is not possible in ALGOL as it stands.

(iii) Algorithms involving multiple branching and sentinels could be better expressed. Consider the situation when a variable or its value can be in three or more possible states (for instance, the value of an integer variable being prime of form 4n+1, prime of form 4n−1, or non-prime, or, to take an example from an algorithm to be used in our ALGOL translator, the type of a variable being **real, integer** (or **string**) or **Boolean**). At one point in the program the state is found and later the program branches according to this state. At present a numerical indicator must be used for the state, for example, taking the values 0, 1 or 2. A branching statement might read in present notation:

if *type* = 0 **then** ··· **else if** *type* = 1 **then** ··· **else** ··· ;

The meaning of this statement could be appreciated much more readily if it were written using the additions:

if *type* = 'real' **then** ··· **else if** *type* = *'integer'* **then** ··· **else** ··· ;

In addition, anywhere a sentinel (i.e. a numerical variable whose value is meaningless as a number but is used to convey information) is used, it would be useful to make it a string variable. When a sentinel has only two values a boolean variable (e.g. one named "number is prime") can be used. This is primarily an improvement to ALGOL in its role as a publication language for algorithms.

We think it would cost virtually nothing to build our additions into a translator. Presumably, on encountering a string a translator encodes it into the internal alpha-