ALGORITHM 93
GENERAL ORDER ARITHMETIC
MILLARD H. PERSTEIN
Control Data Corp., Palo Alto, Calif.

procedure arithmetic (a, b, c, op);
integer a, b, c, op;
comment This procedure will perform different order arithmetic
operations with b and c, putting the result in a. The order of the
operation is given by op. For op = 1 addition is performed. For
op = 2 multiplication, repeated addition, is done. Beyond these
the operations are non-commutative. For op = 3 exponentiation,
repeated multiplication, is done, raising b to the power c. Beyond
these the question of grouping is important. The innermost
implied parentheses are at the right. The hyper-exponent is
always c. For op = 4 tetration, repeated exponentiation, is
done. For op = 5, 6, 7, etc., the procedure performs pentation,
hexation, heptation, etc., respectively.

The routine was originally programmed in FORTRAN for the
Control Data 160 desk-size computer. The original program
was limited to tetration because subroutine recursiveness in
Control Data 160 FORTRAN has been held down to four levels in
the interests of economy.

The input parameter, b, c, and op, must be positive integers,
not zero;
begin own integer d, e, f, drop;
if op = 1 then
begin
a := b + c; go to 1
end if op = 2 then d := 0;
else d := 1; e := c; drop := op - 1;
for f := 1 step 1 until d do
begin
arithmetic (a, b, d, drop);
d := a
end;
1:
end arithmetic

ALGORITHM 94
COMBINATION
JEROME KURTZBERG

procedure combination (J, N, K); value N, K; integer array J; integer N, K;
comment This procedure generates the next combination of N
integers taken K at a time upon being given N, K and the pre-
vious combination. The K integers in the vector J(1) .. J(K)
range in value from 0 to N - 1, and are always monotonically
strictly increasing with respect to themselves in input and
output format. If the vector J is set equal to zero, the first
combination produced is N - K, . . . , N - 1. That initial combina-
tion is also produced after 0, 1, . . . , N - 1, the last value in that
cycle;
begin integer B, L;
B := 1;
mainbody: if J(B) >= B then begin A := J(B) - B - 1;
for L := 1 step 1 until B do J(L) := L + A;
go to exit end;
if B = K then go to initiate;
B := B + 1; go to mainbody;
initiate: for B := 1 step 1 until K do J(B) := N - K + 1 + B;
exit: end combination

ALGORITHM 95
GENERATION OF PARTITIONS IN PART-COUNT
FORM
FRANK STOCKMAL
System Development Corp., Santa Monica, Calif.

procedure partgen (c, N, K, G); integer N, K; integer array c; Boolean G;
comment This procedure operates on a given partition of the
positive integer N into parts <= K, to produce a consequent
partition if one exists. Each partition is represented by the
integers c[1] thru c[K], where c[i] is the number of parts of the
partition equal to the integer i. If entry is made with G = false,
procedure ignores the input array c, sets G = true, and pro-
duces the first partition of N ones. Upon each successive entry
with G = true, a consequent partition is stored in c[1] thru c[K].
For N = KX, the final partition is c[K] = X. For N = KX + r,
1 <= r <= K - 1, final partition is c[K] = X, c[r] = 1. When entry
is made with array c = final partition, c is left unchanged and G
is reset to false;
begin integer a, i, j;
if ~ G then go to first;
j := 2;
a := c[1];
test: if a < j then go to B;
c[i] := 1 + c[i];
c[1] := a - j;
zero: for i := 2 step 1 until j - 1
do c[i] := 0;
go to EXIT;
B: if j = K then go to last;
a := a + j * c[i];
j := j + 1;
go to test;
first: G := true;
c[1] := N;
j := K + 1;
go to zero;
last: G := false;
EXIT: end partgen

ALGORITHM 96
ANCESTOR
ROBERT W. FLOYD
Armour Research Foundation, Chicago, Ill

procedure ancestor (m, n); value n; integer n; Boolean m;

344 Communications of the ACM
**ALGORITHM 97**
**SHORTEST PATH**

ROBERT W. FLOYD

Armour Research Foundation, Chicago, Ill.

**procedure** shortest path \(m, n; \) \(\text{value } n; \text{ integer } m, \) \(\text{array } m; \)

**comment** Initially \(m[i, j] \) is \(\text{true} \) if individual \(i \) is a parent of individual \(j \). At completion, \(m[i, j] \) is \(\text{true} \) if individual \(i \) is an ancestor of individual \(j \). That is, at completion \(m[i, j] \) is \(\text{true} \) if there are \(k, l, \ldots, m[p, j] \) are all \(\text{true} \). Reference: WARSHALL, S. A theorem on Boolean matrices, \(J. ACM \) 9(1962), 11-12;

begin
integer \(i, j, k; \)
for \(i := 1 \) step 1 until \(n \) do
for \(j := 1 \) step 1 until \(n \) do
if \(m[j, i] \) then
for \(k := 1 \) step 1 until \(n \) do
if \(m[i, k] \) then
begin
integer \(s := m[j, i] + m[i, k]; \)
if \(s < m[j, k] \) then \(m[j, k] := s \)
end
shortest path
end

**贡献**

本文中包含的算法在“算法部门”政策声明中所规定的形式。除了ALGOL 60的符号外，ALGOL 60的符号应该被用在“通信”（Communications）杂志上（见“通信”（Communications）杂志，1960年）。建议将本文发送给J. H. Wegstein，National Bureau of Standards，Washington 25, D.C. 算法应该遵循《算法》杂志的参考形式，ALGOL 60的符号应该在一种风格的符号中出现。这可能会导致一些符号看起来与符号符号相同。

**即使**

虽然在整个算法中，文字的贡献者，没有用在“通信”（Communications）杂志上的责任。本书的责任由贡献者的作者，或the Association for Computing Machinery，或the Association for Computing Machinery，或the Association for Computing Machinery，或the contribution of the author and to the Communications issue bearing the algorithm.

**ALGORITHM 98**

**EVALUATION OF DEFINITE COMPLEX LINE INTEGRALS**

JOHN L. PFALTZ

Syracuse University Computing Center, Syracuse, N.Y.

**procedure** COMPLINEINTGRL \(A, B, N, \) \(\text{RSUM} \) \(\text{real } A, B, N; \text{ array } \) \(\text{RSUM} \)

**comment** COMPLINEINTGRL approximates the complex line integral by evaluating the partial Riemann-Stieltjes sum \(S_{n} = \int f(z) \, dz \) where \(a \leq t \leq b \) and \(z(t) \in (z_{i-1}, z_{i}) \). The programmer must provide 1) the procedures \(\text{GAMMA}(T, Z) \) to calculate \(z(i) \) on \(P \), and \(\text{FUNCT}(Z, F) \) to calculate function values, and 2) the end points \(A \) and \(B \) of the parametric interval and \(N \) the number of subintervals into which \([a, b]\) is to be partitioned;

begin
integer \(I; \text{ real } T, \) \(\text{ DELT; \ real } \) \(\text{array } ZT, ZTL, \) \(\text{DELZ, ZK, PART[1:2]; \text{ RSUM[1]} := 0.0; \text{ RSUM[2]} := 0.0; \text{ DELT := (B - A)/N; T := A;}}\)

line: \(\text{GAMMA}(T,Z) ; \)
if \(T = A \) then go to next;
for \(I := 1 \) step 1 until \(2 \) do
begin
\(\text{DELZ}[I] := ZT[I] - ZTL[I]; \) \(\text{end;}\)
for \(I := 1 \) step 1 until \(2 \) do
begin
\(\text{ZK}[I] := \text{ZTL}[I] + \text{DELZ}[I]/2.0; \) \(\text{end;}\)
\(\text{FUNCT}(ZK, FZ); \)
\(\text{PART}[1] := FZ[1] \times \text{DELZ}[1] - FZ[2] \times \text{DELZ}[2]; \)
\(\text{PART}[2] := FZ[1] \times \text{DELZ}[2] + FZ[2] \times \text{DELZ}[1]; \)
for \(I := 1 \) step 1 until \(2 \) do
begin
\(\text{RSSUM}[I] := \text{RSSUM}[I] + \text{PART}[I]; \) \(\text{end;}\)
if \(T < B - (0.25 \times \text{DELT}) \) then go to next else go to exit;

next: for \(I := 1 \) step 1 until \(2 \) do
begin
\(\text{ZTL}[I] := \text{ZT}[I]; \) \(\text{end;}\)
\(T := T + \text{DELT;}\)
go to line;
exit: \(\text{end COMPLINEINTGRL;}\)

**ALGORITHM 99**

**EVALUATION OF JACOBI SYMBOL**

STEPHEN J. GARLAND AND ANTHONY W. KNAPP

Dartmouth College, Hanover, N. H.

**procedure** Jacobi \(n,m,r; \) \(\text{value } n,m; \)
integer \(n, m, r; \)

**comment** Jacobi computes the value of the Jacobi symbol \(n/m \), where \(m \) is odd, by the law of quadratic reciprocity. The parameter \(r \) is assigned one of the values \(-1, 0, \) or \(1 \) if \(m \) is odd. If \(m \) is even, the symbol is undefined and \(r \) is assigned the value \(2 \). For odd \(m \) the routine provides a test of whether \(m \) and \(n \) are relatively prime. The value of \(r \) is 0 if and only if \(m \) and \(n \) have a nontrivial common factor. In the special case where \(m \) is prime, \(r = -1 \) if and only if \(n \) is a quadratic nonresidue of \(m \);

begin
integer \(s; \)
Boolean \(p, q; \)

**Boolean procedure** parity \((x); \) \(\text{value } x; \text{ integer } x; \)

**comment** The value of the function parity is \(\text{true} \) if \(x \) is odd, \(\text{false} \) if \(x \) is even;

begin
parity := \(x \div 2 \times 2 \neq x \)
end parity;

Communications of the ACM 345
if \( \neg \text{parity}(m) \) then begin \( r := 2 \); go to exit end;

\[ p := \text{true}; \]

loop: \( n := n \mod m \times m; \)

\[ q := \text{false}; \]

if \( n \leq 1 \) then go to done;

even: if \( \neg \text{parity}(a) \) then begin

\[ n := n \mod 2; \]

go to even;

end \( n \) now odd;

if \( q \) then if \( \text{parity}((m^2 - 1) \mod 8) \) then \( p := \neg p \);

if \( n = 1 \) then go to done;

\[ \text{even: if } n \text{ odd then begin} \]

\[ q := \neg q; \]

\[ n := n \mod 2; \]

go to loop;

end \( r := \) if \( n = 0 \) then 0 else if \( p \) then 1 else \( -1 \);

exit: end Jacobi

---

**Algorithm 100**

**ADD ITEM TO CHAIN-LINKED LIST**

**PHILIP J. KIVIAT**

United States Steel Corp., Appl. Research Lab., Monroeville, Penn.

**Algorithm 101**

**REMOVE ITEM FROM CHAIN-LINKED LIST**

**PHILIP J. KIVIAT**


**Algorithm 102**

**PERMUTATION IN LExicographical ORDER**

G. F. SCHRACK AND M. SHMRAT

University of Alberta, Calgary, Alberta, Canada
ALGORITHM 103
SIMPSON'S RULE INTEGRATOR
GUY F. KENCIR
UNIVAC Division, Sperry Rand Corp., San Diego, Calif.

procedure SIMPSON (a, b, f, I, i, eps, N);
value a, b, eps, N; integer N;
real a, b, i, eps; real procedure f;
comment This procedure integrates the function f(x) using a
modified Simpson's Rule quadrature formula. The quadrature is
performed over j subintervals of [a,b] forming the total area I.
Convergence in each subinterval of length (b-a)/2^j is indicated
when the relative difference between successive three-point and
five-point area approximations
\[ A_{3,j} = \frac{1}{3} (f(a) + 4f(b) + f(a + b)) \]
\[ A_{5,j} = \frac{1}{5} (f(a) + 4f(b) + 5f(a + b)) \]
is less than or equal to an appropriate proportion of the over-all
tolerance eps (i.e., \( |A_{5,j} - A_{3,j}| / |A_{3,j}| \leq \text{eps}/2^j \) with n \leq N).
SIMPSON will reduce the size of each interval until this condition
is satisfied.

Complete integration over [a,b] is indicated by i = b. A value
a \leq x \leq b is indicates that the integration was terminated, leaving
I the true area under f in that interval. It is recommended that this procedure be used
between known integrand maxima and minima.

begin integer m, n; real d, h; array g[0:4], A[0:2], S[1:N,1:3];
A[0] := (b -- a) X (g[0] + 4 X g[2] + g[4])/2;
g[2] := f((a + b)/2);
g[0] := f(a);
g[4] := f(b);
AA: d := 2Xn; h := (b - a)/4Xd;
BB: if nL = 2X(m + 2) then
end;
begin m := m + 1; i := a + m X (b -- a)/d;
BB: if m = 2X(m + 2) then
begin m := m + 2; n := n - 1; go to BB end
end;
begin A[0] := S[n,1]; g[0] := g[4];
g[2] := S[n,2]; g[4] := S[n,3]; go to AA end
end
CC: end SIMPSON

REMARK ON ALGORITHM 19

RICHARD STECK
Armidge Research Foundation, Chicago 16, Ill.

The for clause of Algorithm 19 should read:

for i := 0 step 1 until b-1 do

With this correction the algorithm was certified on the Armidge Research Foundation UNIVAC 1105.

The recursion formula stated in the comment should read:

\[ G_{n+i} = (n-i) G_{n+i} G_{n+i+1} \]

CERTIFICATION OF ALGORITHM 46
EXPONENTIAL OF A COMPLEX NUMBER (J. R. Herndon, Comm. ACM 4 (Apr., 1961), 178)
A. P. RELPH
Atomic Power Div., The English Electric Co., Whetstone, England

Algorithm 46 was translated using the DEUCE ALGOL compiler, no corrections being required, and gave satisfactory results.

CERTIFICATION OF ALGORITHM 48
LOGARITHM OF A COMPLEX NUMBER (J. R. Herndon, Comm. ACM 4 (Apr., 1961), 179)
A. P. RELPH
Atomic Power Div., The English Electric Co., Whetstone, England

Algorithm 48 was translated using the DEUCE ALGOL compiler, after certain modifications had been incorporated, and then gave satisfactory results.

The original version will fail if a = 0 when the procedure for
arctan is entered. It also assumes that \(-\pi/2 < d < \pi/2\), whereas the
principal value for logarithm of a complex number assumes
\(-\pi < d \leq \pi\).

Incidentally, the ALGOL 60 identifier for natural logarithm is ln,
not log.

The modified procedure is as follows:

procedure LOGC (a,b,c,d); value a,b; real c,d;
comment This procedure computes the number
c + di = log (a + bi).

begin integer m,n, eps;
begin m := sign (a); n := sign (b);
c := sqrt(a X a + b X b);
d := 1.5707963 X (1-m) X (1-n-n X n) + arctan (b/a);
then begin m := 2Xm; n := n + 1;
if n > N then go to CC;
end;
begin m := m + 1; i := a + m X (b -- a)/d;
if m = 2X(m + 2) then
begin m := m + 2; n := n - 1; go to BB end
end;
begin A[0] := S[n,1]; g[0] := g[4];
g[2] := S[n,2]; g[4] := S[n,3]; go to AA end
end
CC: end LOGC

CERTIFICATION OF ALGORITHM 58
MATRIX INVERSION (Donald Cohen, Comm. ACM 4, May 1961)
RICHARD A. CONGER
Yalem Computer Center, St. Louis University, St. Louis, Mo.

Invert was hand-coded in FORTRAN for the IBM 1620. The
following corrections were found necessary:

The statement \( a_{k,1} := a_{k,1} - b_1 X c_k \) should be
\( a_{k,1} := a_{k,1} - b_1 X c_k \).

The statement go to back should be changed to

i := z_k; z_k := z_i; z_i := i; go to back

After these corrections were made, the program was checked by
inverting a 6 X 6 matrix and then inverting the result. The second
result was equal to the original matrix within round-off.
CERTIFICATION OF ALGORITHM 66
INVRS (J. Caffrey, Comm. ACM, July 1961)

JOHN CAFFREY
Palo Alto Unified School District, Palo Alto, California

INVRS was translated using the Burroughs 220 Algebraic Computer (BALcOM) at Stanford University, using 5-digit floating-point arithmetic. The misprint noted by Randell and Broyden (Comm. ACM, Jan. 1962, p. 50) was corrected, and the same example (Wilson's 4 X 4 matrix) was used as a test case. The resulting inverse was:

\[
\begin{bmatrix}
68.0000 & -41.0000 & -17.0000 & 10.0000 \\
25.0000 & 10.0000 & -6.0000 & -3.0000 \\
5.0000 & -3.0000 & -2.0000 & 2.0000 \\
-2.0000 & 2.0000 & 3.0000 & 4.0000
\end{bmatrix}
\]

It may also be useful to note that the determinant of the matrix may be obtained as the successive product of the pivots. That is, if \( h_i \) \((= T(1,1))\) is the \( i \)th pivot of a matrix of order \( n \).

For the above input example,

\[\text{determinant} = 1.0\]

Randell and Broyden's observation concerning the apparent limitation of INVRS to positive definite cases is correct. That is, any nonsingular real symmetric matrix (positive, indefinite, or negative) may be inverted using this algorithm. The original INVRS should therefore be modified as follows:

if pivot = 0 then go to singular;

Randell and Broyden's second example (of order 5) was also used as a test case, with the resulting inverse:

\[
\begin{bmatrix}
-0.0000 & 0.9999 & 0.0000 & 0.0000 & 0.9999 \\
1.5333 & -0.7333 & -0.1333 & 0.7999 & \\
-0.8666 & -1.0666 & -0.5999 & \\
-1.4066 & -0.1999 & 0.2000 & \\
\end{bmatrix}
\]

\[\text{determinant} = -14.999999\]

An attempt to invert the inverse of the 4 X 4 segment of the Hilbert matrix, as presented by Randell (Comm. ACM, Jan. 1962, p. 50), yielded the following results:

\[
\begin{bmatrix}
0.9999 & 0.4999 & 0.3333 & 0.2499 & \\
0.3333 & 0.2499 & 0.1999 & \\
0.1999 & 0.1606 & 0.1428 & \\
\end{bmatrix}
\]

\[\text{determinant} = 6048020.6\]

CERTIFICATION OF ALGORITHM 67
CRAM (J. Caffrey, Comm. ACM 4 (July 1961), 322)

A. P. RELPH
Atomic Power Div., The English Electric Co., Whetstone, England

CRAM was translated using the DEUCE ALGOL compiler with the following corrections:

\[V[i] = S \text{ was changed to } V[i] := S\]

\[f[k,j] = V[k] \text{ was changed to } f[k,j] := V[k]\]

It is quicker not to use the table of the \( C[i] \) in the "load" sequence and instead use the following sequence:

\[
\text{load: m := n X (n-1)/2; for i := 1 step 1 until m do READ (a[i]);}
\]

REMARK ON ALGORITHM 76
SORTING PROCEDURES (Ivan Flores, Comm. ACM 5, Jan. 1962)

B. RANDELL
Atomic Power Div., The English Electric Co., Whetstone, England

The following types of errors have been found in the Sorting Procedures:

1. Procedure declarations not starting with procedure.
2. Bound pair list given with array specification.
3. = used instead of :=, in assignment statements, and in a for clause.
4. A large number of semicolons missing (usually after end).
5. Expressions in bound pair lists in array declarations depending on local variables.
6. Right parentheses missing in some procedure statements.
7. Conditional statement following a then.
8. No declarations for \( A \), or \( z \), which is presumably a misprint.
9. In several procedures attempt is made to use the same identifier for two different quantities, and sometimes to declare an identifier twice in the same block head.
10. In the Presort quadratic selection procedure an array, declared as having two dimensions, is used by a subscripted variable with only one subscript.
11. At one point a subscripted variable is given as an actual parameter corresponding to a formal parameter specified as an array.
12. In several of the procedures, identifiers used as formal parameters are redeclared, and still assumed to be available as parameters.
13. In every procedure \( K \) is given in the specification part, with a parameter, whilst not given in the formal parameter list.
14. No attempt has been made to translate, or even to understand the logic of these procedures. Indeed it is felt that such a grossly inaccurate attempt at ALGOL should never have appeared as an algorithm in the Communications.

CERTIFICATION OF ALGORITHM 77
AVINT (Paul E. Hennion, Comm. ACM 5, Feb. 1962)

VICTOR E. WHITTIER

AVINT was transliterated into BAC-220 (a dialect of ALGOL-58) and was tested on the Burroughs 220 computer. The following minor errors were found:

1. The first statement following label L11 should read:
   \[\text{dif := } 2 \times a \times xarg + b;\]
2. The semicolon (;) at the end of the line beginning with the label L16 should be deleted.
3. There appears to be a confusion between "1" (numeric) and "I" (alphabetic) following label L12. This portion of the program should read:
   \[L12: \text{sum := 0; syl := } \text{xlo} ; \text{ jul := nop - 1; ib := 2};\]
   After making the above corrections the procedure was tested for interpolation, differentiation, and integration using \( e^x \), \( \log X \), and \( \sin X \) in the range \( 1.0 \leq X \leq 5.0 \). Twenty-one values of each of these functions, evenly spaced with respect to \( X \) and accurate to at least 7 significant digits, were tabulated in the above range.
   Then the procedure was tested. The following table indicates approximately the accuracy obtained:

<table>
<thead>
<tr>
<th>Function</th>
<th>Number of Significant Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation</td>
<td>Differentiation</td>
</tr>
<tr>
<td>( e^x )</td>
<td>\text{( \geq 4 )}</td>
</tr>
<tr>
<td>( \log X )</td>
<td>\text{( \geq 4 )}</td>
</tr>
<tr>
<td>( \sin X )</td>
<td>\text{( \geq 4 )}</td>
</tr>
</tbody>
</table>

* Except for interpolation between the first two points in the table.

The above results are quite reasonable in view of the relatively large increment in \( X \). Tests using smaller increments in \( X \) and uneven spacing of \( X \) were also satisfactory.

It was also discovered that for integration the following restrictions must be observed:

1. \( xlo \leq x_1 \) (1).
2. \( xup \geq x_n \) (nop).