```
comment Initially m [i, j] is true if individual i is a parent of
  individual j. At completion, m [i, j] is true if individual i is an
  ancestor of individual j. That is, at completion m[i, j] is true
  if there are k, l, etc. such that initially m[i, k], m[k, l], ···, m[p, j]
  are all 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
m [j, k] := true
end ancestor
```

ALGORITHM 97
SHORTEST PATH

ROBERT W. FLOYD
Armour Research Foundation, Chicago, Ill.

```
procedure shortest path (m,n); value n; integer n; array m;
comment Initially m[i, j] is the length of a direct link from
  point i of a network to point j. If no direct link exists, m [i, j] is
  initially ⏨10. At completion, m [i, j] is the length of the shortest
  path from i to j. If none exists, m [i, j] is ⏨10. Reference: WAR-
  SHALL, S. A theorem on Boolean matrices. J,ACM 9(1962), 11-12;
begin
integer i, j, k; real inf, s; inf := ⏨10;
for i := 1 step 1 until n do
for j := 1 step 1 until n do
if m [j, i] < inf then
for k := 1 step 1 until n do
if m [i, k] < inf then
begin s := m [j, i] + m [i, k];
if s < m [j, k] then m [j, k] := s
end
end shortest path
```

Contributions to this department must be in the form stated in the Algorithms Department policy statement (*Communications*, February, 1960) except that ALGOL 60 notation should be used (see *Communications*, May 1960). Contributions should be sent in duplicate to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D. C. Algorithms should be in the Reference form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department. For the convenience of the printer, please underline words that are delimiters to appear in boldface type.

Although each algorithm has been tested by its contributor, no warranty, expressed or implied, is made by the contributors, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material, and no responsibility 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.

ALGORITHM 98
EVALUATION OF DEFINITE COMPLEX LINE INTEGRALS

JOHN L. PFALTZ
Syracuse University Computing Center, Syracuse, N. Y.

```
procedure COMPLINEINTGRL(A, B, N, RSSUM);
  value A, B, N; real A, B, N; array RSSUM;
comment COMPLINEINTGRL approximates the complex line
  integral by evaluating the partial Riemann-Stieltjes sum
  Σ_{t=1}^{n} f(z_k)[z_t − z_{t−1}] where a ≦ t ≦ b and z_k ∈ (z_{t−1}, z_t). The
  programmer must provide 1) the procedures GAMMA(T, Z) to
  calculate z(t) on Γ, and 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; real T, DELT; real array ZT, ZTL, DELZ,
        ZK, PART[1:2]; RSSUM[1] := 0.0; RSSUM[2] := 0.0;
        DELT := (B − A)/N; T := A;
line:   GAMMA(T, ZT);
        if T = A then go to next;
        for I := 1 step 1 until 2 do
        begin
          DELZ[I] := ZT[I] − ZTL[I]; end;
        for I := 1 step 1 until 2 do
        begin
          ZK[I] := ZTL[I] + DELZ[I]/2.0; end;
        FUNCT(ZK, FZ);
        PART[1] := FZ[1] × DELZ[1] − FZ[2] × DELZ[2];
        PART[2] := FZ[1] × DELZ[2] + FZ[2] × DELZ[1];
        for I := 1 step 1 until 2 do
        begin
          RSSUM[I] := RSSUM[I] + PART[I]; end;
        if T < B − (0.25 × DELT) then go to next else go to
          exit;
next:   for I := 1 step 1 until 2 do
        begin
          ZTL[I] := ZT[I]; end;
        T := T + DELT;
        go to line;
exit:   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); 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 param-
  eter 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); value x; integer x;
    comment The value of the function parity is true if x is
      odd, false if x is even;
    begin
      parity := x ÷ 2 × 2 ≠ x
    end parity;
```

```
  if ¬ parity (m) then begin r := 2;  go to exit end;
  p := true;
  loop:  n := n − n ÷ m × m;
         q := false;
         if n ≦ 1 then go to done;
  even:  if ¬ parity (n) then
         begin
           q := ¬ q;
           n := n ÷ 2;
           go to even
         end n now odd;
         if q then if parity ((m↑2 − 1)÷8) then p := ¬ p;
         if n = 1 then go to done;
         if parity ((m−1) × (n−1) ÷ 4) then p := ¬ p;
         s := m;  m := n;  n := s;  go to loop;
  done:  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.

```
procedure  inlist (t,info,m,list,n,first,flag,addr,listfull);
integer  n,m,first,flag,t;  integer array info,list,addr;
comment  inlist adds the information pair {t,info} to the chain-
  link structured matrix list (i,j), where t is an order key ≧ 0, and
  info(k) an information vector associated with t. info(k) has di-
  mension m, list(i,j) has dimensions (n × (m+3)). flag denotes
  the head and tail of list(i,j), and first contains the address of the
  first (lowest order) entry in list(i,j).  addr(k) is a vector con-
  taining the addresses of available (empty) rows in list(i,j).
  Initialization: list(i,m+2) = flag, for some i ≦ n. If list(i,j) is
  filled exit is to listfull;
begin integer i, j, link1, link2;
0:  if addr [1] = 0;  then go to listfull;  i := 1;
1:  if list [i,1] ≦ t
      then begin if list [i,2] ≠ 0 then begin link1 := m+2;
        link2 := m+3;  go to 2 end;  else begin if
        list [i,m+2] = flag then begin i := flag;
        link1 := m+3;  link2 := m+2;  go to 3 end;
        else begin i := i+1;  go to 1 end end end;
      else begin link1 := m+3;  link2 := m+2 end;
2:  if list [i,link2] ≠ flag
      then begin k := i;  i := list [i,link2];
        if (link2 = m+2 ∧ list [i,1] ≦ t) ∨
           (link2 ≠ m+2 ∧ list [i,1] > t) then go to 4;
        else go to 1 end;
      else begin list [i,link2] := addr [1] end;
3:  j := addr [1];  list [j,link1] := i;
    list [j,link2] := flag;  if link2 = m+2 then
    first := addr [1];  go to 5;
4:  j := addr [1];  list [j,link1] := list [i,link1];
    list [i,link1] := list [k,link2] := addr [1];
    list [j,link2] := i;
5:  list [j,1] := t;  for i := 1 step 1 until m do
    list [j,i+1] := info [i];  for i := 1 step 1 until n−1 do
    addr [i] := addr [i+1];  addr [n] := 0
end  inlist
```

ALGORITHM 101
REMOVE ITEM FROM CHAIN-LINKED LIST

PHILIP J. KIVIAT
United States Steel Corp., Appl. Res. Lab., Monroeville, Penn.

```
procedure  outlist (vector,m,list,n,first,flag,addr);
integer  n,m,first,flag;  integer array vector,list,addr;
comment  outlist removes the first entry (information pair with
  lowest order key) from list(i,j) and puts it in vector(k);
begin integer i;
for i := 1 step 1 until m+1 do vector[i] := list [first,i];
for i := n−1 step −1 until 1 do addr [i+1] := addr [i];
addr [1] := first;
if list [first,m+3] = flag then
  begin list [1,m+2] := flag;  first := 1;
  for i := 1 step 1 until n do addr [i] := i end;
else begin first := list [first,m+3];
list [first,m+2] := flag end;
for i := 1 step 1 until m+3 do list [addr [1], i] := 0
end outlist
```

ALGORITHM 102
PERMUTATION IN LEXICOGRAPHICAL ORDER

G. F. SCHRACK AND M. SHIMRAT
University of Alberta, Calgary, Alberta, Canada

```
procedure  PERMULEX(n,p);
integer  n;  integer array p;
comment  Successive calls of the procedure will generate all
  permutations p of 1,2,3,···,n in lexicographical order. Before the
  first call, the non-local Boolean variable 'flag' must be set to
  true. If after an execution of PERMULEX 'flag' is false,
  additional calls will generate further permutations—if true, all
  permutations have been obtained;
begin integer array q[1:n];  integer i, k, t;  Boolean flag2;
if flag then
  begin for i := 1 step 1 until n do
  p[i] := i;  flag2 := true;  flag := false;
  go to EXIT
  end initialize;
if flag2 then
  begin t := p[n];  p[n] := p[n−1];  p[n−1] := t;
  flag2 := false;  go to EXIT
  end bypass;
flag2 := true; for i := n−2 step −1 until 1 do
  if p[i] < p[i+1] then go to A;
  flag := true;  go to EXIT;
A:  for k := 1 step 1 until n do q[k] := 0;
    for k := i step 1 until n do q[p[k]] := p[k];
    for k := p[i] + 1 step 1 until n do
    if q[k] ≠ 0 then go to B;
B:  p[i] := k; q[k] := 0;
    for k := 1 step 1 until n do
    if q[k] ≠ 0 then begin i := i + 1; p[i] := q[k] end
    else if i ≧ n then go to EXIT;
EXIT:
end PERMULEX
```