ALGORITHM 58
MATRIX INVERSION

DONALD COHEN
Burroughs Corporation, Pasadena, Calif.

```
procedure invert (n) array: (a);
comment matrix inversion by Gauss-Jordan elimination;
    value n;
    array a; integer n;
begin
    array b, c [1:n]; integer i, j, k, ℓ, p;
    integer array z [1:n];
        for j := 1 step 1 until n do z[j] := j;
        for i := 1 step 1 until n do begin
        k := i; y := a[i, i]; ℓ := i - 1; p := i + 1;
        for j := p step 1 until n do begin
        w := a[i, j]; if abs(w) > abs(y) then begin
        k := j; y := w end end;
        for j := 1 step 1 until n do begin
        c[j] := a[j, k]; a[j, k] := a[j, i];
        a[j, i] := -c[j]/y; b[j] := a[i, j] := a[i, j]/y end ;
        a[i, i] := 1/y; j := z[i]; z[i] := z[k]; z[k] := j ;
        for k := 1 step 1 until ℓ, p step 1 until n do
        for j := 1 step 1 until ℓ, p step 1 until n do
        a[k, j] := a[k, i] - b[j] × c[k] end; ℓ := 0 ;
back:   ℓ := ℓ + 1; k := z[ℓ]; if ℓ ≦ n then begin
        for j := ℓ while k ≠ j do begin
        for i := 1 step 1 until n do begin
        w := a[j, i]; a[j, i] := a[k, i]; a[k, i] := w end ;
        go to back end
        end invert.
```

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 $a_{k,j} := a_{k,i} - b_j \times c_k$ *should be*

$$a_{k,j} := a_{k,j} - b_j \times c_k$$

The statement **go to** back *should be changed to*

$$i := z_k;\ z_k := z_j;\ z_j := i;\ \textbf{go to}\ \text{back}$$

After these corrections were made, the program was checked by inverting a 6 × 6 matrix and then inverting the result. The second result was equal to the original matrix within round-off.

CERTIFICATION OF ALGORITHM 58
MATRIX INVERSION [Donald Cohen, *Comm. ACM*, May, 1961]

RICHARD GEORGE*
Particle Accelerator Div., Argonne National Lab., Argonne, Ill.

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

This procedure was programmed in FORTRAN and reduced to machine code mechanically. It was run on the Argonne-built computing machine, GEORGE. A floating-point routine was used which allows maximum accuracy to 31 bits.

There are a number of errors of various types:

(1) There are eight **begin**'s and only seven **end**'s.

(2) The line

$$a[k, j] := a[k, i] - b[j] \times c[k]\ \ \textbf{end};$$

should be

$$a[k, j] := a[k, j] - b[j] \times c[k]\ \ \textbf{end};$$

(3) The permutation of rows of the inverted matrix and permutation of elements of the integer array z must be carried out simultaneously. This algorithm fails to do this, and consequently the matrix at the time of exit from the procedure is left in a permuted condition.

(4) The algorithm permits the statement

$$k := z[l];$$

to be executed even though the declarations place an upper limit of n on the integer array z, and the test for $l \leqq n$ has not yet been made. Obviously, Mr. Cohen's compiling system would allow an out-of-bounds array look-up. One could easily incorporate into an ALGOL compiler a guard against such illicit array references, and therefore the published algorithm might be considered machine dependent.

(5) This algorithm requires $3n^2$ divisions, most of which are unnecessary. By inserting the statement

$$y := 1.0/y;$$

at the proper place, one may accomplish the obvious economy of reducing this to only n divisions plus $2n^2$ multiplications.

(6) If a matrix should be singular (or nearly so), some pivot element will be zero (or very small), and a test should be made, with provision for a jump to *ALARM*, a non-local label.

(7) The identifiers w and y should be declared within this procedure, to avoid trouble.

(8) This algorithm omits calculation of the determinant of the matrix. This could be computed with very little extra effort.

The revised algorithm was then tested on the LGP-30 computer, using ALGOL-30, a small subset of ALGOL. Within the restrictions of this subset, the program worked satisfactorily on test matrices.

REMARK ON ALGORITHM 58
MATRIX INVERSION [Donald Cohen, *Comm. ACM*, May, 1961]

GEORGE STRUBLE
University of Oregon, Eugene, Oregon

For the last seven lines, beginning with $a[k, j] := a[k, i]$, substitute:

```
a[k, j] := a[k, j] − b[j] × c[k] end;
l := 0;
back:   l := l+1;
again:  k := z[l];
if k ≠ l then
  begin for i := 1 step 1 until n do
    begin w := a[l, i];
          a[l, i] := a[k, i];
          a[k, i] := w end;
    z[l] := z[k];
    z[k] := k;
    go to again end;
  else if l ≠ n go to back
end invert
```

REMARK ON ALGORITHM 58
MATRIX INVERSION [D. Cohen, *Comm. ACM*, May 61]

PETER G. BEHRENZ
Matematikmaskinnmänden, Box 6131, Stockholm 6, Sweden

invert was run on FACIT EDB using FACIT-ALGOL 1. Some changes in the procedure had to be made:

1. y and w had to be declared in the procedure-body as **real** y, w;

2. The last part of the procedure starting with $l := 0$; which should interchange the matrix rows did not work correctly, even with the corrections proposed by R. A. Conger [*Comm. ACM*, June 62]. We propose the following code:

```
for l := 1 step 1 until n do begin
k := z[l];  for j := l while k ≠ j do begin
for i := 1 step 1 until n do begin
w := a[j, i];  a[j, i] := a[k, i];  a[k, i] := w end;
i := z[k];  z[k] := z[j];  k := z[j] := i end end end invert
```

If the matrix a is singular, the value of the pivot element y will once be zero or very nearly zero and division by zero would occur in the course of the calculation. It would therefore be advantageous to introduce an empirical tolerance parameter *epsilon* into the procedure.

To calculate the determinant of the matrix a it is only necessary to put three more statements into the code. With these augmentations *invert* should read:

```
procedure invert (n, a, epsilon, determinant);
value n, epsilon;  real epsilon, determinant;
array a;  integer n;
begin real y, w;  integer i, j, k, l, p;
array b, c[1:n];  integer array z[1:n];
determinant := 1;
```

followed by the same code as before until:

```
y := w end end;
determinant := y × determinant;
if k ≠ i then determinant := −determinant;
if abs (y) < epsilon then go to singular;
```

followed by the same code as before with the changes mentioned in the certification by R. A. Conger [*Comm. ACM*, June 62] and the changes given above. *singular* should be a nonlocal label in the main program.