

ALGORITHM 154
COMBINATION IN LEXICOGRAPHICAL ORDER
CHARLES J. MIFSUD
Armour Research Foundation, ECAC Annapolis, Md.

procedure *COMB1* (*n,r,I*); **integer** *n, r*; **integer array** *I*;
comment The distinct combinations of the first *n* integers taken *r* at a time are generated in *I* in lexicographical order starting with an initial combination of the *r* integers 1, 2, ..., *r*. Each call of the procedure, after the first, must have in *I* the previous generated combination. The Boolean variable *first* is nonlocal to *COMB1* and must be **true** before the first call. Thereafter *first* remains **false** until all combinations have been generated. When calling *COMB1* with *I* containing *n - r + 1, n - r + 2, ..., n, I* is left unchanged and *first* is set **true**;
begin integer *s, j*;
if first then **begin for** *j := 1* **step 1** **until** *r* **do**
 I[*j*] := *j*;
 first := **false**; **go to** *EXIT* **end**;
 begin if *I*[*r*] < *n* **then begin** *I*[*r*] := *I*[*r*] + 1; **go to** *EXIT* **end**;
 end;
 for *j := r* **step -1** **until** 2 **do**
 if *I*[*j-1*] < *n - r + j - 1* **then**
 begin *I*[*j-1*] := *I*[*j-1*] + 1;
 for *s := j* **step 1** **until** *r* **do**
 I[*s*] := *I*[*j-1*] + *s - (j-1)*; **go to** *EXIT* **end end**;
 first := **true**;
 EXIT := **end**

ALGORITHM 155
COMBINATION IN ANY ORDER
CHARLES J. MIFSUD

Armour Research Foundation, ECAC Annapolis, Md.

procedure *COMB2* (*m,M,n,r,s,S,TOTAL*); **integer array** *m, M, S*; **integer** *n, r, s, TOTAL*;
comment Each call of *COMB2* generates a distinct combination *S*, (if possible) of the *n* integer values of *J* taken *r* (*r*>1) at a time if *J* consists of *m*[1] integers each equal to *M*[1], and *m*[2] integers each equal to *M*[2], and so on, there being *s* integers available. *TOTAL* must be set to zero before the first call of *COMB2* and thereafter *TOTAL* is increased by one after each new combination is generated. To speed up the machine operation arrange the *s* integers in *M* such that *m*[1] ≥ *m*[2] ≥ ... ≥ *m*[*s*];
begin integer *i, j, t, p*; **own integer array** *J*[1:*n*], *I*[1:*r*]; **own Boolean** *first*;
if *TOTAL* = 0 **then begin**
 t := 1; *p* := 0;
 for *j := 1* **step 1** **until** *s* **do**
 begin *p* := *p* + *m*[*j*];
 for *i := t* **step 1** **until** *p* **do**
 begin *J*[*i*] := *M*[*j*];
 t := *t* + 1 **end end**;
 first := **true** **end**;
1: *COMB1* (*n,r,I*);
 if first **then go to** *EXIT*;
 if *I*[1] = 1 **then go to** 2 **else go to** 3;
2: **for** *j := 2* **step 1** **until** *r* **do**
 if (*J*[*I*[*j*]] = *J*[*I*[*j*]-1]) ∧ (*I*[*j*] > *I*[*j*-1] + 1) **then go to** 1;
 go to 4;
3: **if** *J*[*I*[1]] = *J*[*I*[1]-1] **then go to** 1 **else go to** 2;
4: **for** *j := 1* **step 1** **until** *r* **do**
 S[*j*] := *J*[*I*[*j*]];
 TOTAL := *TOTAL* + 1;
EXIT := **end**



J. WEGSTEIN, Editor

ALGORITHM 156
ALGEBRA OF SETS
CHARLES J. MIFSUD

Armour Research Foundation, ECAC Annapolis, Md.

procedure *INOUT* (*A,n,SUM*); **real array** *A*; **integer** *n*;
real *SUM*;
comment $SUM = \sum_1 A_i - \sum_2 A_i A_j + \sum_3 A_i A_j A_k - \dots \pm A_1 A_2 \dots A_n$ is formed where the symbols $\sum_1, \sum_2, \sum_3, \dots, \sum_{n-1}$ stand for summation of the possible combinations of the numbers A_1, A_2, \dots, A_n taken one, two, three, ..., (*n-1*) at a time;
begin real *j, part, T*; **integer** *i, r*; **integer array** *I*[1:*n*]; **Boolean** *first*;
r := *SUM* := 0; *j* := -1;
B: *first* := **true**; *r* := *r* + 1; *part* := 0;
A: *COMB1* (*n,r,I*);
 if first **then begin** *j* := -1 × *j*; *part* := *j* × *part*;
 SUM := *SUM* + *part*;
 if *r* < *n* **then go to** *B* **else go to** *EXIT* **end**;
 T := 1;
 for *i := 1* **step 1** **until** *r* **do**
 T := *A*[*I*[*i*]] × *T*;
 part := *part* + *T*; **go to** *A*;
EXIT := **end**

ALGORITHM 157
FOURIER SERIES APPROXIMATION
CHARLES J. MIFSUD

Armour Research Foundation, ECAC Annapolis, Md.

procedure *FOURIER* (*N,f,a,b*); **real array** *f, a, b*; **integer** *N*;
comment *Fourier* determines $2N+1$ constants a_p ($p=0,1,\dots,N$), b_p ($p=1,2,\dots,N$) in such a way that the equations $f_n = 1/2a_0 + \sum_{p=1}^N (a_p \cos 2\pi np / (2N+1) + b_p \sin 2\pi np / (2N+1))$ are satisfied, where the f_n are given numbers. The f_n may be thought of as the $2N+1$ values of a function $f(x)$ at the points $x_n = 2\pi n / (2N+1)$. The method used to generate a_p, b_p was formulated by G. Goertzel in "Mathematical Methods for Digital Computers" (John Wiley and Sons, Inc., 1960);
begin real array *S, C*[1:2], *u*[0:2]; **real** *TEMP, pi*;
integer *p, i*;
pi := 3.14159265; *C*[2] := 1; *S*[2] := 0;
C[1] := $\cos(2 \times \pi / (2 \times N + 1))$;
S[1] := $\sin(2 \times \pi / (2 \times N + 1))$;
for *p := 0* **step 1** **until** *N* **do**
 begin *u*[1] := *u*[2] := 0;
 for *i := 2* × *N* **step -1** **until** 1 **do**
 begin *u*[0] := *f*[*i*] + 2 × *C*[2] × *u*[1] - *u*[2];
 u[2] := *u*[1]; *u*[1] := *u*[0] **end**;
 a[*p*] := $2 / (2 \times N + 1) \times (f[0] + u[1] \times C[2] - u[2])$;
 b[*p*] := $2 / (2 \times N + 1) \times u[1] \times S[2]$;
 TEMP := *C*[1] × *C*[2] - *S*[1] × *S*[2];
 S[2] := *C*[1] × *S*[2] + *S*[1] × *C*[2];
 C[2] := *TEMP* **end end**

A contribution to this department must be in the form of an Algorithm, a Certification, or a Remark. Contributions should be sent in duplicate to the Editor and should be written in a style patterned after recent contributions appearing in this department. An algorithm must be written in ALGOL 60 (see *Communications of the ACM*, January 1963) and accompanied by a statement to the Editor indicating that it has been tested and indicating which computer and programming language was used. For the convenience of the printer, contributors are requested to double space material and underline delimiters and logical values that are to appear in boldface type. Whenever feasible, Certifications should include numerical values.

Although each algorithm has been tested by its contributor, no warranty, express or implied, is made by the contributor, 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 158 (ALGORITHM 134, REVISED) EXPONENTIATION OF SERIES

HENRY E. FETTIS

Aeronautical Research Laboratories, Wright-Patterson
Air Force Base, Ohio

procedure *SERIESPWR* (*A,B,P,N*); **value** *A, P, N*;
array *A, B*; **integer** *N*;

comment This procedure calculates the first *N* coefficients *B*[*i*] of the series $g(x) = f(x) \uparrow P$ given the first *N* coefficients of the series

$$f(x) = 1 + \sum A[i] \times x \uparrow i \quad (i=1,2,-,N).$$

P may be any real number. Setting $P := 0$ gives the coefficients for $LN(g(x))$;

begin integer *i, k*;
real *P, S*;
if $P = 0$ **then** *B*[1] = *A*[1];
else *B*[1] := $P \times A$ [1];
for *i* := 2 **step** 1 **until** *N* **do**
begin *S* := 0;
for *k* := 1 **step** 1 **until** *i* - 1 **do**
S := *S* + $(P \times (N-k) - k) \times B$ [*k*] $\times A$ [*N-k*];
B[*i*] := $P \times A$ [*i*] + (*S*/*i*)
end for *i*;
end *SERIESPWR*

ALGORITHM 159 DETERMINANT

DAVID W. DIGBY

Oregon State University, Corvallis, Ore.

real procedure *Determinant* (*X,n*);
value *n*; **integer** *n*; **array** *X*;
comment *Determinant* calculates the determinant of the *n*-by-*n* square matrix *X*, using the combinatorial definition of the determinant. This algorithm is intended as an example of a

recursive procedure which is somewhat less trivial than *Factorial* (Algorithm 33);
begin real *D*; **integer** *i*; **Boolean array** *B*[1:*n*];
procedure *Thread* (*P,e,i*);
value *P, e, i*; **real** *P*; **integer** *e, i*;
if $i > n$ **then** $D := D + P \times (-1) \uparrow e$ **else if** $P \neq 0$ **then**
begin integer *j, f*;
f := 0;
for *j* := *n* **step** -1 **until** 1 **do**
if *B*[*j*] **then** *f* := *f* + 1 **else**
begin
B[*j*] := **true**;
Thread ($P \times X$ [*i,j*], *e+f*, *i+1*);
B[*j*] := **false**;
end of loop;
end of Thread;
for *i* := 1 **step** 1 **until** *n* **do**
B[*i*] := **false**;
D := 0;
Thread (1,0,1);
Determinant := *D*;
end Determinant;

CERTIFICATION OF ALGORITHM 79 DIFFERENCE EXPRESSION COEFFICIENTS

[Thomas Giamo, *Comm. ACM*, Feb. 1962]

EVA S. CLARK

University of California at San Diego, La Jolla, California

The procedure was translated into FORTRAN and run on the CDC 1604. Reasonable accuracy was obtained for $k = 0, 4 \leq n \leq 12$. For increasing *n* and increasing *k*, the accuracy diminished. It was found that the execution time increased rapidly as *n* was increased. For $k = 0$, the following results were obtained:

<i>n</i>	Approximate Number of Machine Operations
4	1.3×10^3
6	6.9×10^3
8	3.8×10^4
10	1.8×10^5
12	8.6×10^5

The author indicated in a letter that the procedure was developed for use with small *n* and small *k*.

CERTIFICATION OF ALGORITHM 96 ANCESTOR [Robert W. Floyd, *Comm. ACM*, June, 1962]

HENRY C. THACHER, JR.*

Argonne National Laboratory, Argonne, Ill.

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

The body of this procedure was tested on the LGP-30 using the Dartmouth translator. After inclosing conditional statements in **begin end** brackets (apparently necessary for this translator), the procedure operated satisfactorily for the following matrices:

n=5, Time: 8'15"
 FTTF FTTT
 FFFT FFFT
 FFFT → FFFT
 FFFT FFFT
 FFFF FFFF

$n = 6$, Time: 13'15"

FTTFFF	FTTTTT
FFFTF	FFFTT
FFFFTF	→ FFFFT
FFFFFF	FFFFFF
FFFFFF	FFFFFF

$n = 9$, Time 31'2"

FTTTTTFFF	FTTTTTTTT
FFFFFFFFF	FFFFTTTTF
FFFTTTTTF	FFFTTTTTT
FFFFFFFFF	FFFFFFTTT
FFFFTTTTF	→ FFFFFTTTF
FFFFFFFTF	FFFFFFFTF
FFFFFFFTF	FFFFFFFTF
FFFFFFFTF	FFFFFFFTF
FFFFFFFTF	FFFFFFFTF
FFFFFFFTF	FFFFFFFTF

The correctness of these results was confirmed by inspection of the network diagrams.

CERTIFICATIONS OF ALGORITHMS 117 and 118
MAGIC SQUARE (ODD AND EVEN ORDERS)

[D. M. Collison, *Comm. ACM*, Aug. 1962]

K. M. BOSWORTH
I.C.T. Ltd., Blyth Road, Hayes, Middlesex, England

The statement within the **Booleon procedure beta** should be changed from

$x[r,a] := \text{if } h \text{ then } [nn-a \times n+r] \text{ else } (a \times n+1-r);$
to
 $x[r,a] := \text{if } h \text{ then } (nn-a \times n+r) \text{ else } (a \times n+1-r);$

The procedures were then tested on magic squares of order 3 to 17 inclusive without fault.

REMARK ON ALGORITHM 133
RANDOM [Peter G. Behrenz, *Comm. ACM* 11, Nov. 1962]

DONALD L. LAUGHLIN
Missouri School of Mines and Metallurgy, Rolla, Missouri

Algorithm 133 was translated into FORTRAN II for the IBM 1620 and run successfully. The starting value was changed to 21 348 759 609 and significant results followed.

For $N = 500$ and 1000, the resulting values were: 0.4990157688, 0.4986269653 and 0.3318717863, 0.3290401482.

NATIONAL JOINT COMPUTER CONFERENCES

SPRING JCC May 21-23, 1963 DETROIT

FALL JCC November 12-14, 1963 LAS VEGAS

SPRING JCC May 26-28, 1964 WASHINGTON

Sponsored by the American Federation of Information Processing Societies (AFIPS)

Note on the Proof of the Non-existence of a Phrase Structure Grammar for ALGOL 60

PETER J. BROWN

University of North Carolina, Chapel Hill, N. C.

The proof of the non-existence of a phrase structure grammar for ALGOL 60 by Robert W. Floyd [*Comm. ACM* 5 (Sept. 1962)] depends on the assumption that a syntactically correct ALGOL program must be a block. The concept of "program" is defined ambiguously in the ALGOL Report, as pointed out by Naur [1], but it is generally accepted that a program is defined as a self-contained statement. If this definition is taken, Floyd's proof becomes incomplete in that it ignores the fact that the following are syntactically correct ALGOL programs:

- (1) **begin; end**
- (2) **begin end**
- (3) (dummy statement)

The proof can be easily extended to include these cases. Writing (as in the notation of Floyd's proof) P_i as the concatenation of the strings $QR^{(i)}ST^{(i)}U$ and taking P_i as the string

begin real $x^{(n)}$; $x^{(n)} := x^{(n)}$ end

it is apparent that if P_0 is the string

begin; end

then Q is the string **begin**

R " " " **real $x^{(n)}$**
 S " " " ;
 T " " " $x^{(n)} := x^{(n)}$
 U " " " **end**

and thus it follows that P_2 is the string

begin real $x^{(n)}$ real $x^{(n)}$; $x^{(n)} := x^{(2n)} := x^{(n)}$ end

which is not a syntactically correct program.

If P_0 is the string

begin end

then since S cannot be a null string (it is defined as a proper substring of RST), S must be either the string

begin

or the string

end.

In either case, Q , R , T and U can be derived and it follows that P_2 is not a syntactically correct program.

Lastly, it is obvious that P_0 cannot be the null string since S cannot be null. Hence P_0 cannot be any of the substrings of P_1 that are programs but not blocks.

A further small point about the proof might be worthy of note. The assumption that all variables must be declared implies that the version of ALGOL being considered contains no standard functions (such as carriage return, π). This latter assumption can be made without loss of generality.

REFERENCE

1. NAUR, P. Remark concerning the Definition of a Program. ALGOL Bulletin No. 10.