1971

Introduction to ALGOL 60 for those who have used other programming languages

A. Nico Habermann
Carnegie Mellon University

Introduction to ALGOL 60 for those who have used other programming languages

by A. N. Habermann

September, 1971

Carnegie-Mellon University
Department of Computer Science
Pittsburgh, Pa. 15213

Introduction to ALGOL 60
for those who have used other programming languages

by A.N. Habermann

Including Problems and the Revised Report

0. Introduction

The idea to write an introduction to ALGOL 60 for people who have programming experience in other languages stems from Dr. A.J. Perlis. He suggested a series of such introductions to various languages. This introduction was written having in mind the FORTRAN or BASIC programmer. The APL programmer may have some more difficulty to understand the features of ALGOL 60 as explained here with respect to the control statements. This introduction to ALGOL 60 does not claim to be complete. The idea was rather to present to the experienced programmer the flavor of ALGOL 60 and illustrate its major features. The author's preference and taste certainly shows in the presented material in the sense that goto statements and switches are not discussed at all.

The problems are closely related to the presentation and are numbered accordingly. The Revised Report on ALGOL 60 gives the programmer the exact description of the language and it is recommended that he use it when there is any doubt about permissability or correctness of program writing. The presentation is machine-independent; the programmer should read the appropriate manual for implementation details.

1. Data structures

1.1 Constant

1.2 Variable

1.3 Expression

1.4 Array

1.5 Procedure

1.1 Constant

type is expressed by representation

integer constants	21, -273
real constants	2.718, $3.14_{10}{}^{-2}$, ${}_{10}{}^{-3}$, .1138
boolean constants	true and false
string constants	"any sequence of symbols"

1.2 Variable

A variable is represented by a name (official term: identifier).

A name does not reflect the type.

A variable is declared to be of certain type in a declaration.

A declaration consists of a declarator followed by a list of variable names and is terminated by a semicolon.

Note: transitions to a new line or a new page or next card have no meaning in ALGOL 60 as delimiting symbols.

```
integer  i, j, k;  real x, y, z;
     Boolean  a, b, c; integer n;
```

A variable can get a value by means of an "assignment statement"

```
i := j + 1;    x := 2 * y + z;   a := true
```

1.3 Expressions

1.3.1 Arithmetic expressions

unary operators - +

binary operators in order of priority ↑
* / ÷
+ -

a - b * c ↑ 2

Operations of equal priority are evaluated from left to right.

Two arithmetic operators should not be written in succession.

a * - b is incorrect.

Parentheses overrule the priority:

a * (b + c) a * (-b).

Note: * / have equal priority.

Hence, 24/6 * 4 evaluates to 16 (and not to 1).

The type of an arithmetic expression depends usually on the types of its operands.

p + q, p - q, p * q	are of type integer if both operands are of type integer, and of type real if at least one of the operands is of type real.
p / q, p ↑ q	are of type real if the operands are of type real or integer.
p ÷ q (integer division)	is only defined for two operands of type integer; the result is also of type integer.

When a real expression is assigned to an integer variable, its value is rounded to the nearest integer $i := x + 0.7$ where $x = 2.6$ yields $i = 3$.

1.3.2 Relations

Relational operators $< \leq = \neq \geq >$ are binary operators all of the same priority, which is lower than that of + and -.

The operands are of type integer or real, the result is of type Boolean.

$i > k + 1$ $\quad x \geq 0$ $\quad n < i \uparrow (j + 2 * k)$

N.B. $x < y \leq z$ is illegal, because the type of $x < y$ is Boolean and $\leq$ requires 2 operands of type integer or real.

1.3.3 Boolean expressions

operators in priority order

$\neg$	not	(unary operator)
$\wedge$	and	(binary operators)
$\vee$	or	
$\rightarrow$	imp	
$\equiv$	eqv	

$\neg a$ $\quad b \wedge \neg c \vee x < y + z$

The type of a Boolean expression is Boolean.

The value of a Boolean expression is either true or false.

table

a	b	$\neg$ a	a $\wedge$ b	a $\vee$ b	a $\rightarrow$ b	a $\equiv$ b
t	t	f	t	t	t	t
t	f	f	f	t	f	f
f	t	t	f	t	t	f
f	f	t	f	f	t	t

The value of a Boolean expression can be assigned to a Boolean variable

$a := b \wedge c \equiv x > 0$

1.4 Array

An array is an ordered set of variables.

The array is represented by a name.

The elements are represented as subscripted variables:

```
v [4]     M [ 3,7]     v [i + k]     M [ i + 1, j - 2].
```

The subscript expression must be of type integer or real.

Its value is rounded to the nearest integer value.

All subscripted variables of an array are of the same type. This type is not reflected in the representation and therefore an appropriate array declaration must be given.

```
integer array    v[6:24];
real array       M[3:101, 7:9];
boolean array    p, q, r [0:7];
```

A subscripted variable can be used in places in the program where a variable could be used.

```
v[i + 3] * ( v[i + 1] ↑2 - v[i + 2]) + 2 * i * v[i]

p[3] ∧ a ≡ q[6] ∨ v          p[i] := a ∧ y ≤ z + 1

v[i + j] := 2 * k + 1
```

1.5 Procedure

A procedure is a piece of program represented by a name. Execution of a procedure may require some parameters. These are given in the so called actual parameter list.

```
x := SIN(0.75); y := LN (2.718);  READ(x);  PRINT(i);

i := MAX(i,j);    SWAP (x,y)
```

Some procedures are considered to be available to any ALGOL 60 program. These procedures are said to be in the ALGOL library.

Library procedures are

ABS	LN	and usually
ENTIER	EXP	READ
SIN	SIGN	WRITE
COS	SQRT	PRINT
ARCTAN		although their use varies from one implementation to another.

A programmer can also write his own procedures (like MAX and SWAP). In such cases a procedure declaration is required. This is discussed in section 4.

2. Control statements

Declarations are followed by one or more "statements". Statements are executed in sequence. Statements are separated from each other by a semicolon (since new line, next card and new page have no meaning).

```
i: = 3; v[i] :=5;  i := i + 1;  v[i] := v[i -1] + 2 *
v[i - 1] ↑2
```

2.1 Repetition statements

```
for i:= expr1 step expr2 until expr3 do S
```

i must be a variable of type real or integer (usually the latter). (Don't forget that i must occur in a declaration!).

expr1, expr2, expr3 must be expressions of type real or integer. (a constant or a variable is also a form of expression).

```
for i := 50 step 1 until 90 do v[i] := 0;
```

The elements 50 through 90 of vector v are set to zero.

```
for i := 0 step 1 until n do
    for j := 0 step 1 until i - 1 do  a[i,j] := a[j,i]
```

```
s:=1;  for  i := 1 step s until 20 do  s :=s + 1
```

The statement s := s + 1 is executed for i = 1, 3, 6, 10, 15 successively. The final value of s = 6.

```
n := - 14;
for i := 1 step - 2 until n - 1 do  n := n + 2
```

The statement n := n + 2 is executed for i = 1, -1, -3, -5, -7 successively. The final value of n = -4.

2.2 Compound statement

Any sequence of statements can be grouped together into one statement

by surrounding the sequence with the bracket pair

```
    begin  ...  end
```

```
for  i := 3 step 1 until 10 do
   begin  k := v[i];
          v[i] := v[18 - i];
          v[18 - i] := k
   end
```

2.3 Conditional repetition

for i := expr while BE do S

i must be a variable of type integer or real.

expr must be an expression of type real or integer.

BE must be an expression of type Boolean.

Statement S is executed repeatedly as long as BE is found true. The value of expr is assigned to variable i every time BE is evaluated.

```
i := 12;
for i := i + 1 while i ≤ 20 do  v[i] := SQRT (v[i])
```

A very useful conditional repetition statement has been implemented in more recent ALGOL 60 systems.

while BE do S

BE must be an expression of type Boolean.

If this statement is not available in the implementation you work with (it is not in ALGOL 60 officially), it can be programmed using the for while statement with a dummy assignment.

```
for a := a while BE do S
```

Let a and b be positive integers. The greatest common divisor of a and b is computed by

```
r := 1;
    while r ≠ 0 do        (or for a := a while r ≠ 0 do)
    begin
        q:= a ÷ b;
        r:= a - q * b;
        a:= b;
        b:= r
    end;
gcd:= a
```

2.4 Conditional statement

if BE then S

BE must be an expression of type Boolean.

Statement S is only executed if BE is true.

Restriction: S may not be itself a conditional statement. This rules out the sequence ... then if ...

This restriction is not a serious one, because any statement can be transformed into a compound statement by surrounding it with begin end.

```
if i < 20 then begin if v[i] < 0 then v[i] := -v[i] end
```

```
for i := 12 step 1 until 36 do
  if v[i] ≠ 0 then begin PRINT(i); PRINT (v[i]) end
```

This statement prints the index and value of the elements unequal zero.

2.5 Selective conditional statement

if BE then S1 else S2

BE must be an expression of type Boolean. Either statement S1 or statement S2 is executed depending on whether BE is true or false.

Restriction: Statement S1 may not be a conditional statement, nor a repetition statement. Statement S2 may be any kind of statement.

As a thumb rule one had better avoid always the sequences ... then if ... and ... then for ...

The restrictions are not serious, since the problem can be avoided by using a compound statement.

```
if n > 5 then
begin for i := 5 step 1 until n do
        if v[i] = i then v[i] := 0
end
```

Suppose it is known that array v has positive elements numbered from 0 through 512. The value of the largest element is computed by

```
max := 0;
for i := 0 step 1 until 512 do
    if  max < v[i] then max := v[i]
```

3. Blockstructure

A block has the form

begin < declarations > ; < statements > end

The names declared in this block are said to be "locals" of this block.

A block is a statement.

A whole ALGOL program has the form of a block.

Declarations are terminated by a semicolon.

A statement is only terminated by a semicolon if it is not followed by else or end.

A block differs from a compound statement in that it contains declarations. When a block is entered, the declared names are added to the ones declared in surrounding blocks. When the end of a block is passed, the names declared in this block are deleted. Thus, the names exist only as long as the block is executed.

```
begin integer i, j; real array A, B, C [0: 10, 0: 10];
for i := 0 step 1 until 10 do
for j := 0 step 1 until 10 do  READ (A[i,j]);
for i := 0 step 1 until 10 do
for j := 0 step 1 until 10 do  READ (B[i,j]);
for i := 0 step 1 until 10 do
for j := 0 step 1 until 10 do
  begin integer k; real sum;
        sum := 0;
        for k := 0 step 1 until 10 do
              sum := sum + A[i,k] * B[k,j];
        C[i,j] := sum
  end;
```

```
for i := 0 step 1 until 10 do
for j := 0 step 1 until 10 do  PRINT (C[i,j])
end
```

It is allowed to program lower- and upperbound of an array declaration in the form of arithmetic expressions with program variables.

```
real array A[0:n], B[n:2 * n + 1];
```

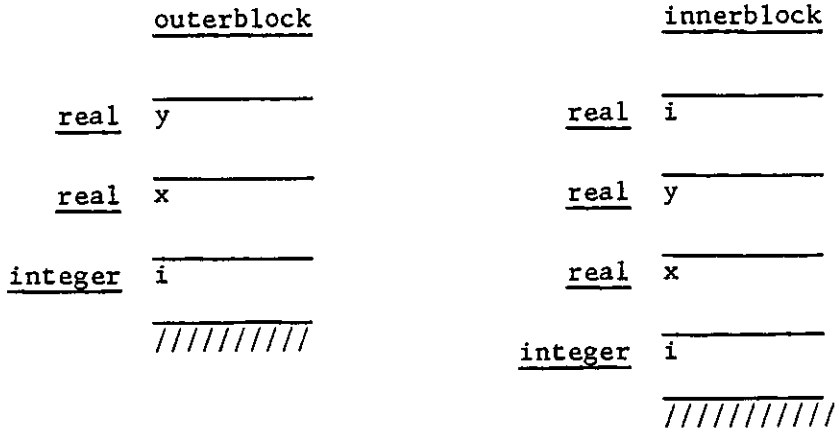
But the value of n must be defined when this declaration is processed. This implies that n cannot be declared in the same block as arrays A and B, since all declarations of a block come before any of its statements and so,no value can have been assigned to n in this block when the array declaration is processed. This problem is solved by using an outerblock:

```
begin integer n;  READ(n);
   begin real array A[0:n], B[n: 2 * n + 1];
         --- --- ---
         --- --- ---
   end
end
```

It is allowed to declare a name in an inner block that already was declared in an outer block.

```
begin integer i; real x,y;
      READ(x); READ(y); i := 3;
      if x > y then
       begin real i;
             i := x;
             x := y;
             y := i;
       end;
      i := i + 1
end
```

When the statement i: = i + 1 is executed, the real variable i does not exist any more. The list of defined names is in the



The location of a name is found by searching the namelist from the top down for the first occurrence of that name. Thus, in the innerblock real i is found and in the outerblock integer i.

4. Procedures

4.1 In the matrix multiplication example the statement to read in arrays A and B are very similar. A procedure to read in any array with boundpair list 0:10, 0:10 is

```
procedure MREAD(W);  real array W;
 begin integer i, j;
       for i := 0 step 1 until 10 do
       for j := 0 step 1 until 10 do   READ (W[i,j])
 end;
```

The first line is the so called procedure heading. It consists of the declarator procedure, the procedure name, the (list of) formal parameter(s) and the so called specification of the parameter(s). The procedure heading is followed by the so called procedure body which must be a statement and which is usually a block.

Procedure MREAD can easily be modified into the more general procedure SQMREAD which reads any square matrix:

```
procedure SQMREAD (W, a, z); array W; integer a, z;
  begin integer i, j;
        for i := a step 1 until z do
        for j := a step 1 until z do  READ (W[i,j])
  end;
```

A procedure is used in a program in the form of a procedure call. In the matrix multiplication example the statements to read arrays A and B could be replaced by the procedure calls:

```
SQMREAD (A, 0, 10);
SQMREAD (B, 0, 10)
```

if the declaration of procedure SQMREAD was placed in the program following the declaration of the arrays A, B and C.

4.2 Type procedures

Type procedures serve to compute the value of a function.

```
y:= SIN(1 - x); c:= 2 * MAX(a,b) + 1;
  a:= MAX(MAX(x,MAX(y,z)), MAX(u,v))
```

If the used type procedure is not in the ALGOL library, it must be properly declared in the program.

```
real procedure MAX(x,y); real x,y;
   begin
         if x > y then MAX:= x else MAX:= y
   end;

integer procedure NORM(v,n); real array v; integer n;
begin  integer i; real sum;
       sum:= 0;
       for i:= 1 step 1 until n do
           sum:= sum + v[i] * v[i];
       NORM:= SQRT (sum)
end;

boolean procedure PRIME (x); integer x;
begin integer i; boolean pr;
      pr:= x - 2 * (x ÷ 2) ≠ 0;
      i:= 1;
      for i:= i + 2 while i ↑2 ≤ x and pr do
      · if x - i * (x ÷ i) = 0 then pr:= false;
      PRIME:= pr
end;
```

The procedure name of a type procedure is used in the procedure

body as a kind of local variable to which the procedure result is assigned. The procedure name may only be used as a local of the body left of the assignment operator:= (or ←). It would for instance lead to incorrect programming when in the last procedure the boolean variable pr were replaced by PRIME because of the expression

```
i ↑2 ≤ x and pr
```

It is allowed that a non-local variable is used in a procedure body. If a value is assigned to such a non-local variable the procedure is said to have side effects. These may be very useful, but one should realize the consequences.

```
begin integer i;
      integer procedure P(x); integer x;
       begin P:= x; x:= x + 1 end;
       i:= 4;  i:= P(i) * i
end
```

The last value assigned to i in this program is 20.

5. Recursion

The rule for constructing names (identifiers) in ALGOL 60 is given in terms of a so called recursive definition.

<identifier>::= <letter>|<identifier> <letter>|<identifier> <digit>

The symbol | is read as "or".

From this definition one can derive that an identifier is a non empty sequence of letters and digits of which the first is a letter. The definition is called "recursive" because the term identifier is defined in terms of itself.

There is a class of functions that can be defined recursively.

For instance $f(a,n) \equiv a^n$ $a \geq 0$, $n \geq 1$ is defined by $f(a,1) = a$ and $f(a,n) = a * f(a,n-1)$

A recursive function can be computed by means of a recursive procedure i.e., a procedure that calls itself in its body.

```
real procedure POWER(a,n); real a; integer n;
    begin
        if n = 1 then POWER:= a
        else POWER := a * POWER(a, n-1)
    end;
real procedure SUM(v,n); real array v; integer n;
    begin
        if n = 1 then SUM:= v[1]
        else SUM:= v[n] + SUM (v, n-1)
    end;
```

The coefficients of Pascal's triangle fulfill the relations

$$C_0^n = 1, \quad C_n^n = 1 \qquad \text{for } n \geq 0$$

$$C_i^n = C_{i-1}^{n-1} + C_i^{n-1} \qquad \text{for } n \geq 0,\ 1 \leq i \leq n-1$$

```
integer procedure PCOEF(n,i); integer n,i;
begin
      if i = 0 or i = n then PCOEF:=1
      else if i ≥ 1 and i ≤ n-1
           then PCOEF:= PCOEF(n-1, i-1) + PCOEF(n-1, i)
           else PCOEF:= 0
end;
```

6. Call by name and call by value

In the last example the formal parameters n and i are used several times in the procedure body. To be more precise: the values of n and i are used several times.

The ALGOL 60 rule for using formal parameters is: when a procedure body is executed, the substituted actual parameter is evaluated every time when the corresponding formal parameter is encountered. This means that on account of the call

PCOEF (a ↑ 2 + 1, 3)

the expression a ↑ 2 + 1 is evaluated every time when the value of n is needed.

This reevaluation of the actual parameter is clearly not needed in the given example. It could easily be avoided by using local variables in the procedure body:

```
integer procedure PCOEF(n,i); integer n,i;
  begin integer locn, loci;
                locn:= n; loci:= i;
        if loci = 0 or  loci = locn then COEF := 1
        else if loci ≥1 and loci ≤ locn -1
             then PCOEF:= PCOEF(locn -1, loci - 1) + PCOEF(locn -1, loci)
             else  PCOEF:= 0
  end;
```

It is, however, not necessary to program such an optimization explicitly with local variables. The programmer has the option of indicating in the declaration of a procedure that a particular parameter has to be evaluated only once. It is said that such a parameter is called by value. The way to do this is to add to the procedure heading

a so called value list, in which the parameters are specified that are "called by value."

```
integer procedure PCOEF (n,i); value n, i; integer n,i;
begin
      if i = 0 or i = n then PCOEF:= 1
      else if i ≥ 1 and i ≤ n-1
          then PCOEF:=PCOEF(n-1, i-1) + PCOEF (n-1, i)
          else PCOEF:= 0
end;
```

Formal parameters called by value should be considered as local variables of the body which will be initialized when the procedure is called with the value of the corresponding actual parameters.

If a parameter is not listed in the value list, it is understood that the corresponding actual parameter is substituted for the formal parameter (and not just its once computed value). This use of a parameter is refered to as "call by name".

To clarify the difference of call by name and call by value this example:

```
begin integer i; integer array v[1:100]; integer a,b;
      integer procedure VSUM(u) ; value u; integer u;
      begin
          integer s; s:= 0;
          for i:= 1 step 1 until 100 do s:= s + u;
          VSUM:= s
      end;
```

```
integer procedure NSUM(u); integer u;
begin
        integer S; S:= 0;
        for i:= 1 step 1 until 100 do S:= S + u;
        NSUM:= S
end;
for i:= 1 step 1 until 100 do  v[i]:= i;
i:= 5;
a:= VSUM(v[i]);
b:= NSUM(v[i])
end
```

When VSUM is called, v[5] is evaluated and u gets (only once) the value 5. Variable a gets the value 500. When NSUM is executed v[i] is substituted for u and the result is that b gets the value 5050.

Epilogue.

Various concepts of ALGOL 60 and several details have been left out of this introduction. The most important ones left out are: goto statements, switches, forlists, multiple assignments. One can find those in other descriptions of the language as for instance in the ALGOL 60 manual for the Univac 1108 or the PDP 10.

A complete and accurate description of ALGOL 60 is found in the

Revised Report on the Programming Language ALGOL 60
P.Naur (ed).
published in Rosen's book,
The Communications of the ACM (Jan 63)
and various other places.

It should be stated that in cases where this introduction deviates from the Report, the latter should be considered as the correct description.

Bibliography on ALGOL 60

Manual

[1] PDP-10 Algol Manual or UNIVAC 1108 ALGO

Definition

[2] Naur, P. (ed.) Revised Report on the Algorithmic Language ALGOL 60. Comm.ACM 6 (Jan 63).

[3] Knuth, D. E. The remaining trouble spots in ALGOL 60. Comm.ACM 10 (Oct. 67).

[4] Abrahams, P. W. A final solution to the dangling else of ALGOL 60 and related languages. Comm.ACM 9 (Sept 66).

[5] Knuth, D.E., Merner, J.N. ALGOL 60 Confidential. CACM, Vol. 4, 1961.

Philosophy

[6] Perlis, A. J. The synthesis of algorithmic systems. J.ACM 14 (Jan 1967).

History-Bibliography

[7] Bemer, R. W. A politico-social history of ALGOL. Annual Review In Automatic Programming, 5 Pergamon Press, 1969.

[8] Sammet, Jean Programming Languages: History and Fundamentals, Prentice-Hall, 1969.

Introductory

[9] Bottenbruch, H. Structure and use of ALGOL 60. J.ACM 9 (Apr 62).

[10] Higman, B. What everybody should know about ALGOL. Computer Journal 6 (1963) p. 50.

[11] Ekman, T. and Froberg, C. Introduction to ALGOL programming. Oxford University Press, (1967).

[12] Dijkstra, E. W. A Primer of ALGOL 60 Programming, Academic Press, London, 1962.

Implementation

[13] Evans, A. — An ALGOL 60 Compiler. Annual Review in Automatic Programming, 4. Pergamon Press (1964).

[14] Randell, B. and Russell, L.J. — ALGOL 60 Implementation. Academic Press, (1964), 418 pp.

[15] Dijkstra, E. W. — "Making A Translator for ALGOL 60," Annual Review of Automatic Programming, Vol. 3., MacMillan, 1963, pp. 347-356.

Extensions

[16] Wirth, N. — A Generalization of ALGOL. Comm.ACM 6 (Sept 63).

[17] Perlis, A.J. and Iturriaga, R. — An extension to ALGOL for manipulating formulae. Comm.ACM 7 (Feb 64).

[18] Wirth, N. and Weber, H. — EULER: A generalization of ALGOL and its formal definition. Comm. ACM 9 (Jan, Feb. 66).

[19] Wirth, N. and Hoare, C.A.R. — A contribution ot the development of ALGOL. Comm.ACM 9 (June 66).

[20] Dahl, O.J. and Nygaard, K. — SIMULA - An ALGOL-based simulation language. Comm.ACM 9 (Sept 66).

1. Data structures

1.1 Constants

integer constants	216	1	-273	+1296
real constants	3.1415&2		2.718&-4	
	+10.10&-4		-12.24&+3	
	42.1516		-0.07236	
	.541778		+.0321927	
	.23&-2		-.032&8	
	&5		-&-7	

Which of the following list represent numbers in ALGOL 60?

3.1&0.5 2. 2&2 3.5.6 3.14&(-2)

1.2 Variables

types: integer real Boolean

Examples of names: low, spring, found, case1, case2, switch1to3

Are the following names correct identifiers in ALGOL 60?:

onethrunine	a6
1thru9	a(6)
bow&arrow	a[6]

1.3 Expressions

```
begin integer i,k; real a,b,c; integer
      bow, arrow; real target; Boolean hit;
      a := 3;  b := a+1;  c := b+2.3&-1;
      PRINT(a); PRINT(b); PRINT(c);
      i := a;  k := b/a*c;  hit := a < b and c ≥ 2;
      PRINT(i); PRINT(k); PRINT(hit);
      hit := hit or b < 1 ≤ a - 2.4/(b*a)*c↑2 ≡ hit;
      PRINT(hit)
end
```

The values assigned to the variables are successively

a: 3.0
b: 4.0
c: 4.23
i: 3
k: 6
hit: true true

Which values are assigned to the variables in the following assignment statements?

```
begin
    integer i,j,k; real a,b,c; Boolean p,q;
       a := 3.1415&+1;
       b := 2.718&-1;
       c := 1.01;
       p := a < b and c > 10*b↑2;
       q := a-b < 3*c or ¬ p;
       i := a+b; j := a+c; k := b+c;
       i := j+2*a; j := i-k*c↑2; k := b*(i-1)-a;
       p := i > j and j > k ≡ i> k;
       q := a/12*b↑2 < c+1/(b/(2*c+1))
end
```

2. Control statements

2.1 Counting repetition

(1) to read in the elements of a matrix stored rowwise on an input file

```
begin integer i,j; real array A[1: 100, 1:100];
      for i := 1 step 1 until 100 do
          for j := 1 step 1 until 100 do  READ(A[i,j])
      - - -
      - - -
```

(2) to zero out the upper triangle of a matrix

```
begin integer i,j; real array A[1: 100, 1: 100];
      for i := 1 step 1 until 100 do
          for j := i+1 step 1 until 100 do  A[i,j] := 0
      - - -
      - - -
```

(3) Horner's scheme for computing the polynomial

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

is to start with $y = a_n$ and repeat n times

$$y := y*x+a_i \quad \text{for } i = n-1, n-2, \ldots, 0$$

```
begin integer i; real array A[0:20]; real x,y;
      for i := 0 step 1 until 20 do READ(A[i]);
      READ(x);
      y := A[20];
      for i := 19 step -1 until 0 do  y := y*x+A[i]
      - - -
      - - -
```

(4) Write a piece of program that computes the sum of the elements on the main diagonal of a given square matrix of 144 elements.

(5) Write a piece of program that assigns to a boolean variable "none" the value true or false according to whether none of the elements of a given vector v of 25 components is zero or not.

(6) Write a piece of program that computes the inner product of two given vectors u and v of 80 components each.

(7) Write a piece of program that computes the length of a given vector of 45 components.

(8) Write a piece of program that gets as input a positive integer less than 1,000,000 and prints successively the digits of the octal representation of this number.

(9) The coefficients of an expansion of $(1+x)^n$ are given by $C[n,0] = 1$ for $n \geq 0$, $C[n,k] = \frac{n}{k} * C[n-1,k-1]$ for $1 \leq k \leq n$. Write a program that computes C[24,13].

2.2 Compound statements

(1) To transpose a square matrix A of 169 elements

```
for i := 1 step 1 until 13 do
   for j := 1 step 1 until i-1 do
      begin
            x := A[i,j];
            A[i,j] := A[j,i];
            A[j,i] := x
      end
```

(2) The sequence of Fibonacci is defined by

$F(0) = 0$, $F(1) = 1$ and $F(n) = F(n-1) + F(n-2)$ for $n \geq 2$

to compute the 25th Fibonacci number

```
begin integer a, b, f, i; a := 0; b := 1;
      for i := 2 step 1 until 25 do
         begin f := a+b;
               b := a;
               a := f
         end
end
```

(3) Matrix A represents 15 vectors of 20 components each; to compute a matrix D that contains the distances between all pairs of vectors in A.

```
begin integer i,j,k; real s; real array A[1:15, 1:20], D[1:15,1:15];
      for i := 1 step 1 until 15 do
         for j := 1 step 1 until 20 do READ(A[i,j]);
      for i := 1 step 1 until 15 do
         begin D[i,i] := 0;
               for j := i+1 step 1 until 15 do
                  begin s := 0;
                        for k := 1 step 1 until 20 do
                            s := s + (A[i,k] - A[j,k]) ↑ 2;
                        s := SQRT(s); D[i,j] := s; D[j,i] := s
                  end
         end
end
```

(4) Matrix A represents 45 vectors of 12 components each. Write a piece of program that assigns to the Boolean variable "sphere" the value true or false according to whether all 45 vectors are equal in length or not.

(5) Write a piece of program that multiplies matrices A[0:24,0:9] and B[0:9,0:11] and stores the result into matrix C of proper size.

(6) A number sequence is given by s[0] = 0, s[n] = n+s[n-1] for n ≥ 1. Write a program that computes s[36].

2.3 Conditional repetition

(1) A root a of the equation f(x) = x can be found by means of successive approximation if in the neighborhood of x = a ABS(f'(x)) ≤ k < 1. The approximation starts with a value x_0 in that neighborhood and a new approximation is computed from the last according to $x_n = f(x_{n-1})$. One can prove using the first mean value theorem that the sequence x_n converges to a.

$F(x) \equiv 0.01(x^4 + 2x^3 - 3x^2 - 4)$

F(x) intersects the x-axis between x=1 and x=2

```
begin real a, newx, x; x := 1; newx := 1.5;
      while ABS(newx-x) > 0.0005 do
      begin x := newx;
            newx := ((x+2)*x-3)*x↑2*0.01-0.04
      end
end
```

(2) Queues $q_1, q_2, \ldots, q_9$ are stored in array A[1:1000]. The first element of q_i is A[i]. If A[k] = n, it means that A[n] is the next element of the queue to which A[k] belongs. The last element of a queue has the value zero, unused elements of A have the value -1.

To delete the last element of queue q_i (assuming the proper declaration and initializations)

```
k := i;  next := i;
while A[next] > 0 do begin k := next; next := A[k] end;
A[k] := 0; A[next] := -1
```

(3) To assign the value true to Boolean variable "zero" if and only if vector v[1:2048] has at least one element equal zero, and, if so, the index of the first element = 0.

```
i := 0;  zero := false;
while i < 2048 and ¬ zero do
begin i := i+1; zero := v[i] = 0 end
```

(4) Write a program that gets a real number x as input and computes n terms of the expansion 1-x + x↑2/2!-x↑3/3!+... until a term with absolute value ≤ 0.0005 is found.

(5) Newton's method to approximate a solution "a" of f(x) = 0 is based on the computation of the sequence $x_0, x_1, x_2, \ldots$ where x_o is a point in the neighborhood of point a and $x_{n+1} = x_n - f(x_n)/f'(x_n)$. Write a program to approximate 6↑1/3 with an accuracy of 0.001 applying Newton's method on $f(x) = x^3-6$.

(6) Write a program that gets as input a positive integer n less than 1000 and assigns to the Boolean variable "prime" the value true or false depending on whether or not n is a prime number.

(7) Write a program that computes the largest Fibonacci number less than 8000.

(8) Write a program that joins all the queues of the example presented earlier into one queue q_1 leaving out the first elements $q_2, q_3, \ldots, q_9$.

(9) Integral (f(x),a,b) is computed by successive approximation.

The first approximation is

```
i(0) := (0.5*f(a) + 0.5*f(b))*(b-a)
```

Approximation i(k+1) is obtained from i(k) by taking the mid-points of every sub interval and adding the approximation of all intervals. So,

```
i(1) := 0.5*(f(a) + f(m))*(b-a)/2 + 0.5*(f(m) + f(b))*(b-a)/2
```

or `i(1) := i(0)/2 + f(m)*(b-a)/2` where `m = (a+b)/2`

```
i(2) := 0.5*(f(a) + f(am))*(b-a)/4 +
        0.5*(f(am + f(m))*(b-a)/4 +
        0.5*(f(m) + f(mb))*(b-a)/4 +
        0.5*(f(mb) + f(b))*(b-a)/4
```

or

```
i2  := i(1)/2 + (f(am) + f(mb))*(b-a)/4
```

where am is the midpoint of segment (a,m) and bm of segment (m,b). Write a program that computes Integral(x*sin(x),0,1) until two approximations differ less than 0.001

2.4 Conditional statement

(1) Vector v[1:100] has positive elements. To compute the value of the largest element

```
begin integer i; real max; real array v[1:100];
      max := 0;
   for i := 1 step 1 until 100 do
            if max < v[i] then max := v[i]

      - - -
      - - -
end
```

(2) a[1:50] and b[1:50] are sets of integer numbers. The sets are called "disjunct" if for all i,j = (1,...,50) either a[i] < b[j] or a[i] > b[j]. To assign to Boolean variable "disjunct" the value true if and only if sets a and b are disjunct. (Assume proper declarations and initialization.)

```
mina := a[1]; maxa := a[1]; minb := b[1]; maxb := b[1];
for i := 2 step 1 until 50 do
   begin  if mina > a[i] then mina := a[i];
          if minb > b[i] then minb := b[i];
          if maxa < a[i] then maxa := a[i];
          if maxb < b[i] then maxb := b[i]
   end;
disjunct := maxa < minb or maxb < mina
```

(3) v is an ordered set of 512 numbers. To find out up until which element the values are in ascending order (assume proper declarations and initialization)

```
i := 1; ascending := true;
while i ≤ 512 and ascending do
   begin i := i+1;
         if v[i] < v[i-1]
         then begin i := i-1; ascending := false end
   end
```

(4) v is a set of 100 "names". A "name" is represented as a positive integer less than 100,000. Write a piece of program to arrange the names in alphabetical order, i.e., in ascending order.

(5) v is a set of 729 integer numbers. Write a piece of program that computes the minimum value of these numbers and how many elements of v are equal to that minimum.

(6) v[1:120] is a set of real numbers. The "diameter" of a set is MAX(v[i] - v[j]) for all i,j = (1,...,120). Write a piece of program that computes the diameter of v.

(7) Matrix A represents a set of 15 vectors of 6 elements each and matrix B represents a set of 12 vectors of 6 elements each. The "distance" between A and B is the minimum of all the distances between a vector of A and one of B. Write a piece of program that computes the distance between sets A and B.

2.5 Selective conditional statement

(1) The maximum of four variables a, b, c and d can be computed in one statement

```
if a > b
then begin if a > c
           then begin if a > d then max := a
                              else max := d
                end
           else if c > d then max := c
                         else max := d
     end
else if b > c
     then begin if b > d then max := b
                         else max := d
          end
     else if c > d then max := c
                   else max := d
```

(although the following sequence is simpler:

```
max := a;
if max < b then max := b;
if max < c then max := c;
if max < d then max := d)
```

(2) A group of 75 students got scores on a progress test with a scale ranging from 1 to 100. The results are regrouped into four categories: excellent, good, average, insufficient. The boundaries are at 90, 70 and 50. To program how many of each category there are:

```
begin integer excellent, good, average, insufficient;
      integer i, score;
      for i := 1 step 1 until 75 do
         begin
               READ(score);
               if score ≥ 90 then excellent := excellent + 1
               else if score ≥ 70 then good := good + 1
                    else if score ≥ 50 then average := average + 1
                         else insufficient := insufficient + 1
         end;
      PRINT(excellent); PRINT(good); PRINT(average); PRINT(insuffici
end
```

(3) Does the following program compute correctly the greatest common divisor for any pair of integer numbers (a,b)?

```
begin integer a,b,gcd;
      READ(a); READ(b);
      while b ≠ 0 do
      if a > b then a := a-b
               else b := b-a;
      gcd := a; PRINT(gcd)
end
```

(4) Let f(x) be a continuous function on interval [a,b] and let f(x) intersect the x-axis exactly once in that interval. The point of intersection is approximated by shrinking the interval to 1/2, 1/4, 1/8, ... of its size successively, preserving only that part in which the zero point lies. Write a piece of program that solves the equation $4x^4 + 3x^3 - 2x^2 - 1 = 0$ with this method.

(5) Array A[1:25] is considered as a circular storage area for a queue q (i.e., each element A[i] has a successor; the successor of A[25] is A[1]). Elements are added at the front of q and elements are deleted from the rear of q. Write pieces of program to add to q an element just read in and to delete an element from q, assuming that the variables "front" and "rear" point to the correct elements of A.

(6) Someone wrote in an ALGOL 60 program

```
begin integer i,j; integer array a[1:100]; Boolean stop;
      for i := 1 step 1 until 100 do READ (a[i]);
      stop := false; i := 0;
      while ¬ stop do
      if i ≤ 100 and a[i] > 0
      then begin PRINT(a[i]); i := i+1 end
      else stop := true
end
```

His program worked all right if he supplied it an input file with at least one 0 or negative number, but it gave a runtime error for an input file with only positive numbers. A friend suggested to replace the if statement by the following statement:

```
if i ≤ 100
then begin if a[i] > 0
           then begin PRINT(a[i]); i := i+1 end
           else stop := true
     end
else stop := true
```

and this cured the problem! Try to explain why.

3. Block structure

(1) The first number on an input file represents the number of components of the vectors and is followed by the components of two vectors. To write a program that computes the inner product

```
begin integer n;
      READ(n);
      begin real array u,v[1:n]; integer i; real inprod;
            for i := 1 step 1 until n do READ(u[i]);
            for i := 1 step 1 until n do
            begin
                  READ(v[i]);
                  inprod := inprod + v[i]*u[i]
            end;
            - - -
            - - -
            - - -
      end
end
```

(2) To program the linear transformation v := A*v where v is a vector of n components and A is the n*n transformation matrix. Blockstructure is used to limit the scope of intermediate variables.

```
begin integer n; READ(n);
   begin real array v[1:n], A[1:n, 1:n];
   begin real array u[1:n];
      begin integer i;
            for i := 1 step 1 until n do READ (u[i]);
      end;
      begin integer i,j;
            for i := 1 step 1 until n do
            for j := 1 step 1 until n do READ(A[i,j])
      end;
      begin integer i;
            for i := 1 step 1 until n do
            begin integer j; real sum;
                  sum := 0;
                  for j := 1 step 1 until n do sum := sum + A[i,j]*u[j];
                  v[i] := sum
            end
      end
   end
   end
end
```

(3) To write a program that solves a set of linear inhomogeneous equations of which the matrix is triangular with a non-zero main diagonal, i.e.,

to solve $A*x = e$

where $a_{ij} = 0$ for all $1 \leq i < j \leq n$ and

$a_{ii} \neq 0$ for all $1 \leq i \leq n$.

Since it is triangular, it pays to store the matrix into a one dimensional array B[1:n*(n+1) ÷ 2] to avoid a waste of storage. The general mapping rule is A[i,j] ≡ B[i*(i-1) ÷ 2+j]

```
begin integer n; READ(n);
   begin real array B[1:n*(n+1) ÷ 2],x,C[1:n];
      begin integer i, upb; upb := n*(n+1) ÷2;
            for i := 1 step 1 until upb do READ(B[i]);
            for i := 1 step 1 until n do READ(C[i])
      end;
      begin integer i;
            for i := 1 step 1 until n do
            begin
                  x[i] := C[i] / B[i*(i+1) ÷ 2];
               begin integer k;
                     for k := i+1 step 1 until n do
                        begin integer m; m := k*(k-1) ÷ 2+i
                              C[k] := C[k] - B[m] * x[i];
                              B[m] := 0
                        end
            end end
      end
   end
end
```

(4) Write a program that computes the coefficients of the product of two polynomials. The input file contains the index of the first polynomial followed by its coefficients, followed by the index of the second polynomial and its coefficients.

(5) C is a vector of n elements. Matrix A is a n*n matrix of which only the elements a[i-1,i] for $2 \leq i \leq n$, a[i,i] for $1 \leq i \leq n$ and a[i,i+1] for $1 \leq i \leq n-1$ are not equal to zero. Write a program to solve the set of linear equations

$$A*x = c$$

The input file contains successively n, the 3n-2 non-zero elements of A and the elements of c.

(6) Array A[0:10,000] is used to store variable size segments of at least two array elements. The first contains the link, i.e., a pointer to the segment next in order of ascending array address (zero for the last segment); the second contains the segment length in number of array elements. A[0] points to the first segment. Segments are added and deleted and so, after a while, unused areas are scattered through array A. Write a program that moves the segments to one end of array A (and updates the links properly) such that only one free area results at the other end of array A.

(7) The Boolean product and Boolean sum of two integers represented as binary numbers is computed by operating on corresponding bits of those numbers according to the following tables:

Boolean product	Boolean sum
0 * 0 = 0	0 + 0 = 0
0 * 1 = 0	0 + 1 = 1
1 * 0 = 0	1 + 0 = 1
1 * 1 = 1	1 + 1 = 0

Thus, 101101 * 011001 = 001001

101101 + 011001 = 110100

The Boolean multiplication is coded on the input tape as 1 and the Boolean addition as 0. Write a program that gets as input the operation code followed by two (decimal) integer numbers $\leq$ 1,000,000, that converts the numbers into binary representation, carries out the indicated operation and prints the sequence of resulting bits.

(8) Write a program that prints out in how many ways, and how, one can pay an amount of n cents (n $\leq$ 100,000) in dollars, quarters, dimes, nickels and cents.

(9) Write a program that prints the triples (a,b,c) of Pythagoras (i.e., the relation between a, b and c is a↑2 + b↑2 = c↑2), for $1 \leq a < b \leq 100$.

4. Procedures

(1) To write a Boolean procedure that gets the value true if and only if a given square matrix is orthonormal.

```
Boolean procedure ORTH(A,n); value n; real array A; integer n;
begin integer i,j; Boolean ok;
      real procedure INPROD(rowi, rowj, n);
           value rowi, rowj, n; integer rowi, rowj, n;
           begin integer i; real sum; sum := 0;
                 for i := 1 step 1 until n do
                    sum := sum + A[rowi, i] * A[rowj,i];
                 INPROD := sum
           end;
      ok := true; i := 0;
      for i := i +1 while i ≤ n and ok do
           ok := INPROD(i,i,n) = 1;
      i := 0;
      if ok then
      begin for i := i+1 while i ≤ n and ok do
           begin j := i;
                 for j := j+1 while j ≤ n and ok do
                          ok := INPROD(i,j,n) = 0
           end
      end;
   ORTH := ok
end;
```

(2) A "push-down stack" is a list of elements to which elements are added and from which elements are deleted at one end of the list called the stack top. Let array A[0:1000] be used as storage area for a stack and let integer variable "top" point to the top

of the stack. A stack element consists of one or more elements of array A, the topmost element contains the length in number of array elements. To write a procedure that adds a stack element stored in v[1:n] to the stack.

```
procedure PUSH(v,n); value n; real array v; integer n;
if top + n ≥ 1000 then WRITE("stack overflow")
else begin integer i;
           for i := 1 step 1 until n do A[top+i] := v[i];
           top := top+n+1;
           A[top] := n
     end;
```

(3) To write a procedure that computes e↑(-x) with an accuracy of 0.0005 and counts the number of terms needed to achieve this accuracy

```
real procedure EPOW(x,n); value x; real x; integer n;
begin real term, sum; integer sign;
      term := 1; sum := 1; sign := 1; n := 1;
      while ABS(term) > 0.005 do
         begin sign := -sign;
               term := sign*term*x/n;
                       sum := sum + term;
               n := n+1
         end;
      EPOW := sum
end
```

(4) One can also pass a function (procedure) name as parameter. To write a procedure that uses the method of bisection to approximate the zero point of a function in a given interval with a given accuracy

```
real procedure ZERO(f,a,b,acc); value acc;
     real a,b,acc; real procedure f;
begin real m;
      if f(a) > 0 then begin m := a; a := b; b := m end;
      while b-a > acc do
      begin m := (b-a)/2;
            if f(m) < 0 then a := m else b := m
      end;
      ZERO := m
end;
```

Suppose the program in which procedure ZERO is declared contains also the declaration

```
real procedure F(x); value x; real x;
     begin F := (2*x+1) * x↑2 - 2 end;
```

The program could use the procedures for instance in the statement

```
y := ZERO(F,0,1,0.0005) or
y := ZERO(SIN,-.5,.5,0.00001)
```

(5) The sequence of Fibonacci numbers is defined by

$$F(0) = 0,\ F(1) = 1,\ F(n) = F(n-1) + F(n-2) \text{ for } n \geq 2.$$

Write a recursive procedure FIB(n) which computes F(n).

(6) Write a procedure F(exp,x,n) that tabulates the values of an expression represented by exp for a set of n arguments represented by x.

(7) Write procedure POP which deletes an element from the stack (see problem 2).

(8) What is the result of the following program?:

```
begin integer i,j,k;
      procedure P(F); procedure F;
         begin integer a;
               procedure Q; begin i := i+1 end;
                  j := j+1;
               if j = 1 then P(Q);
                  F;
                  a := i+k;
                  PRINT(a)
         end;
      procedure S; begin k := k+1 end;
   i := 0; j := 0; k := 0;
   P(S)
end
```

(9) Palindromes. A palindrome is a vector V of values such that V = XY where X = reversal of Y. E.g., 110011. Write a Boolean function that determines if a vector is a palindrome. Write another which determines if a vector consists of a list of palindromes; e.g., 110110.

(10) How well do you understand call-by-name and call-by-value?

```
begin real A,B;
   real procedure INCV(X); value X;real X;
        begin X←X+1; INCV←X end;
   real procedure INCN(X);real X;
        begin X←X+1; INCN←X end;
   real procedure ADDV(Y);value Y;real Y;
        ADDV←Y+Y;
   real procedure ADDN(Y);real Y;
        ADDN←Y+Y;

   A←1; B←ADDV(INCV(A));
   comment A IS NOW ------, B IS NOW -----;

   A←1; B←ADDV(INCN(A));
   comment A IS NOW ------, B IS NOW -----;

   A←1; B←ADDN(INCV(A));
   comment A IS NOW ------,

   A←1; B←ADDN(INCN(A));
   comment A IS NOW ------, B IS NOW -----;
end;
```

Miscellaneous

(1) Sieve of Eratosthenes

Eratosthenes' method to list the prime numbers less than 1000 is the following one:

Set up the sequence 1,2,3,...,1000.

Scratch out 1.

Repeat:

Scratch out all multiples of the first number not yet scratched out (for the first time this number is 2) until to the right of this number all numbers are scratched out.

Write a program that lists the prime numbers less than 1000 in ascending order using the sieve of Eratosthenes.

(2) Continued fractions

let $q_1 = 1/(1+1)$

$q_2 = 1/(1 + 1/(1+1))$ etc.

as i → infinity, $q_1 \rightarrow q = 0.6183 \ldots$

Write a procedure PHI(n) that will return the value q_n.
Write two versions of PHI, one recursive and one iterative.

(3) The QUEEN's problem

To place on the chessboard 8 queens in such a way that no queen can capture another queen in one move. Write a program that produces all solutions of this problem (there are 92 solutions).

(4) Write a procedure SWAP(x,y) that exchanges the values of x and y. Check that it works for SWAP(i,a[i])and SWAP(a[i],i).

(5) 32 locations numbered from 0 through 31 are arranged in a circle (i.e., location 0 follows location 31). The locations are to be filled with either 0 or 1 in such a way that all numbers from 0 through 31 are represented exactly once by interpreting all possible contiguous sets of 5 locations as the representation of a binary number. Write a program that produces all solutions of this problem.

(6) Tower of Hanoi

Given three pins and a pack of n disks on the first pin. The disks are stacked in order of descending diameter. Move the disks to pin 2 (possibly using pin 3 as intermediate storage) so that (1) the disks end up in the same order on pin 2 as they were on pin 1, (2) at no time a larger disk is on top of a smaller one and (3) only one disk at a time is moved. Write a program that solves this problem for an arbitrary number of disks.

(7) (J. Weizenbaum)

Write a program that, for given positive n, determines the smallest number z that can be decomposed into the sum of two n-th powers in at least two non-trivial ways.

For instance: $n = 3 \quad z = 1729 = 1\uparrow 3 + 12\uparrow 3$

$= 9\uparrow 3 + 10\uparrow 3$

$n = 4 \quad z = 635318657 = 59\uparrow 4 + 158\uparrow 4$

$= 133\uparrow 4 + 134\uparrow 4$