A COURSE OF
ALGOL 60 PROGRAMMING

with special reference to the DASK ALGOL system

Second edition

by

Peter Naur

REGNECENTRALEN, COPENHAGEN

1961

CONTENTS.

INTRODUCTION.

The difficulty of learning the algorithmic language ALGOL 60 may be both under and over-estimated. While it is true that a few hours of study under suitable supervision may place the student in a position to express himself intelligently in a basic part of the language, it is clear that a complete mastery of all the possibilities of the language will require considerably more study. On the other hand the feeling of despair which may seize the student who for the first time tries to acquaint himself with the ALGOL 60 report is caused largely by the special character of this report. In fact, being designed to present as concise and complete a description of the language as possible, the ALGOL 60 report cannot be expected to act as a well balanced first introduction as well.

The purpose of the present course is to act as a guide for the student who wishes to acquire a thorough knowledge of the language and some facility in expressing himself in it. Since thoroughness is aimed at it seems obvious from the outset that the course must be based firmly on the ALGOL 60 report itself. For this reason the course itself basically consists only of a set of directions in how to read the ALGOL 60 report and a set of accompanying exercises. Only occasionally have special notes, dealing with particular points in the ALGOL 60 report, been added. Thus it is hoped that having worked through the directions given in the present course, the student will be in a position to understand the ALGOL 60 report and to use it as his standard reference.

Since the course was written primarily for students of the DASK ALGOL translator system the special characteristics of this system are explained and used. This gives the added advantage over a pure reference language course that conventions for input and output are available. On the other hand since the DASK ALGOL representation in its appearance lies very close to the reference language much of the material presented will probably be of more general interest.

Because of the special character of the course the student who is completely unprepared will need an informal introduction to the language. Danish readers may use one of the following articles for this purpose:

W. Heise: ALGOL - et internationalt sprog for elektronregnemaskiner. Ingeniøren nr. 17, 1. sept. 1959 (this article is somewhat out of date, being based on a preliminary version of the language, but will serve as introduction all the same).

P. Naur: ALGOL - det internationale sprog til at beskrive logiske og numeriske processer. Nordisk Matematisk Tidsskrift, Bind 8 (1960) 117.

The course is divided into consecutively numbered points. Within each of these points additional notes and problems may appear. Section numbers will refer to the sections of the ALGOL 60 report. References to A MANUAL OF THE DASK ALGOL LANGUAGE will be written as for example MANUAL section 7.1.2. For each point there is left space open for the student to note the time required to work that point.

THE COURSE.

1. Read through section 1, not including 1.1. (Time   min.)

2. Read through the same section, this time noticing particularly the following concepts:

    arithmetic expressions
    assignment statements
    statements
    labels
    compound statements
    declarations
    blocks
    programs (Time   min.)

3. Read carefully through the problem description given in Appendix 1. Try to recognize instances of some of the concepts listed in point 2. (Time   min.)

3 Problem 1. Which of the concepts listed in 2 do not appear in the problem description of Appendix 1?

4. Study section 1.1 Formalism for syntactic description. (Time   min.)

4 Note 1. The meaning of the syntactic formulae may be further explained by saying that words enclosed in the bracket < >, like <ab>, denote classes whose members are sequences of basic symbols. Class designations of this kind are found in any description of a language. For describing ordinary natural languages designations like word, verb, noun, are used. This of course introduces the logical difficulty that a clear destinction between the language described and the language used for description (the meta-language) is not made (the designation verb is itself a word, but not a verb). This difficulty is avoided in the description of ALGOL by introducing the special mark < > for metalinguistic classes.

The fact that the syntactic rules of ALGOL may be described fully and conveniently by means of the very simple formalism of section 1.1 is of course simply a consequence of the way the language has been defined.

4 Problem 1. Half of the following sequences are values of <ab> as defined in section 1.1, the rest are not. Find those which are.

1.	213	5.	[2.73(	9.	(a + b)
2.	[([	6.	((2(4	10.	(1234567
3.	[12(34	7.	[27(3(((	11.	[22(2(3)
4.	[00	8.	((a	12.	(987(65

(Time min.)

5. Read once carefully through sections 2, 2.1, 2.2.1, 2.2.2, 2.3, including the footnote 1 to section 2.1. Do not try to learn this section by heart.

5 Note 1. Note the following special features of the DASK ALGOL representation:
The DASK ALGOL alphabet includes æ, Æ, ø, Ø.
÷ is not included in DASK ALGOL.
For ↑ DASK ALGOL uses ⅄
⊃ is not included in DASK ALGOL
For ¬ DASK ALGOL uses -,
For ‘ and ’ DASK ALGOL uses { and }
For Boolean DASK ALGOL uses boolean
The third form of comment is permitted only in the restricted form
end <any sequence of digits or letters> (Time min.)

5 Problem 1. Some of the following characters or groups of characters represent basic ALGOL 60 symbols, others do not. Using sections 2 - 2.3 as reference, find those which do.

1.	a7	5.	⅄	9.	end
2.	function	6.	x	10.	go
3.	value	7.	:=	11.	until
4.	:	8.	=:	12.	=>

(Time min.)

5 Problem 2. Use the comment conventions to contract the following sequences as much as syntactically possible:

1. a:=b+3 ; comment Now comes the inner loop ; V: PW:=n ;
2. begin comment This is executed whenever q≤7 ; if PQ=0 then go to W ;
3. Q[n]:=n+7 end section 2 else go to WW
4. tu:=vu/2 end block V and end block sub V ;

(Time min.)

6. Study section 6: 8-channel punch tape code and flexowriter keyboard.
(Time min.)

6 Problem 1. For each of the delimiters which is not an underlined word, find out how it will be typed using the DASK ALGOL keyboard. Assuming that the previous case shift is unknown, find the number of keys to be depressed for each of the delimiters. Arrange the delimiters in groups according to the number of keys to be depressed and find the number of delimiters in each group. (Time min.)

7. Study sections 2.4.1 - 2.4.3.

7 Note 1. In DASK ALGOL only the first six characters of an identifier will be recognized. Thus, although identifiers of any length may be used, two identifiers, to be different, must differ in one or more of the first six characters.

7 Note 2. The complete list of reserved identifiers of DASK ALGOL is given in MANUAL section 7.4.

7 Note 3. The sentence in section 2.4.3 on the same identifier denoting different quantities implies that at any one place in an ALGOL program one cannot have an identifier denoting, say, both a simple quantity (a number) and a matrix (an array of numbers). This restriction is not obvious since it is always possible to recognize array identifiers by the following bracket [ ]. (Time min.)

7 Problem 1. Some of the following sequences of characters can be used as identifiers, others cannot. Mark those which can.

1.	begin	5.	P7.2	9.	7VPQ
2.	a×v	6.	Start value	10.	V̲7
3.	4711	7.	number	11.	a29v3
4.	ppp3	8.	Q(2)	12.	epsilon

(Time min.)

8. Read section 2.5.1 - 2.5.4. (Time min.)

8 Note 1. In DASK ALGOL numbers must be confined to the following ranges

$$- 524288 \leq \text{integer} \leq 524287$$
$$2.94_{10}-39 \leq \text{abs(real)} \leq 3.40_{10}38$$

8 Problem 1. Write numbers having the same values as the following, but which do not include an exponent part.

1.	$+7.293_{10}8$	3.	$_{10}+3$	5.	$-_{10}-6$
2.	$98.12_{10}+2$	4.	$-.1834_{10}-5$	6.	$-4.8_{10}3$

(Time min.)

8 Problem 2. The values given by the following numbers may, in some cases, be expressed more economically by using a number with an exponent part. Show where this is the case.

1.	17000	3.	-0.00134	5.	-0.0020041298
2.	1000	4.	1.0024	6.	170

(Time min.)

8 Problem 3. Some of the following sequences of characters represent numbers, some do not. Mark those which do.

1.	-.0 08	5.	-17.2.30	9.	13.411 732
2.	$+13.47_{10}+18$	6.	±4.2	10.	$2.48_{10}n$
3.	$4 \times {}_{10}-2$	7.	$-88_{10}-7$	11.	× 643.2
4.	(16.20)	8.	$1,24_{10}3$	12.	$12._{10}8$

(Time min.)

9. The sections 2.6 and 2.7 may be skipped for the moment. Read section 2.8. Continue to read section 3 up to and including section 3.1.3 leaving out, however, anything dealing with subscripts or arrays.

9 Note 1. A recursive definition is a definition which uses the defined object itself as a part of it.

9 Note 2. The definition in section 3.1.1 of a simple variable is unnecessarily complicated since the construction <variable identifier> is completely equivalent with <simple variable>. The formulation given was chosen because it was considered desirable that there exist a <variable identifier> analogous to <array identifier>, <procedure identifier>, and <switch identifier>. (Time min.)

9 Problem 1. Which of the examples of section 3.1.2 denote simple variables? (Time min.)

10. Study section 5.1.1 - 5.1.3 on type declarations.

10 Note 1. Remember that DASK ALGOL writes boolean. (Time min.)

10 Problem 1. Some of the following sequences denote type declarations, some do not. Mark those which do.

1. integer q10 q11, h
2. integer
3. boolean integer
4. integer a5, 7
5. real number, HH
6. integer 2a4b, L2, k2
7. own boolean true
8. integer K2, (v)
9. real k ; B
10. integer kappa, Kappa
11. real 2.34
12. real STUFF

(Time min.)

11. Read sections 3.3.1, 3.3.2, the first paragraph of 3.3.3, and 3.3.4 - 3.3.5.2, leaving out, however, anything dealing with function designators, if clauses, and subscripted variables. (Time min.)

11 Note 1. DASK ALGOL does not include ÷.

11 Example 1. The proof that a given construction is an ALGOL 60 arithmetic expression is equivalent to showing that the construction may be formed through applications of the rules of section 3.3.1. Thus for example the construction

$$a \times (b + c \times d \uparrow e \uparrow f) \times g$$

is proved to be an expression through the following steps:

Primaries:	a	b	c	d	e	f	g
Factors:	a	b	c	d			g
and therefore also:		d ↑ e					
and again:		d ↑ e ↑ f					
Terms:	a	b	c				
and therefore also:		c × d ↑ e ↑ f					
Simple arithmetic expressions:		b					
and therefore also:		b + c × d ↑ e ↑ f					

Arithmetic expression:	$b + c \times d \uparrow e \uparrow f$
Primary:	$(b + c \times d \uparrow e \uparrow f)$
Factor:	$(b + c \times d \uparrow e \uparrow f)$
Term:	$a \times (b + c \times d \uparrow e \uparrow f)$
and therefore also:	$a \times (b + c \times d \uparrow e \uparrow f) \times g$
Simple arithmetic expression:	$a \times (b + c \times d \uparrow e \uparrow f) \times g$
Arithmetic expression:	$a \times (b + c \times d \uparrow e \uparrow f) \times g$

Through the analysis we have had occasion to introduce the following numbers of syntactic units:

Primaries:	8
Factors:	8
Terms:	6
Simple arithmetic expressions:	3
Arithmetic expressions:	2

(Time min.)

11 Problem 1. Analyze in the same way as in the previous example the construction of each of the following arithmetic expressions and find the number of different syntactic units in each case.

1. $((P))$
2. $-Q/R \times (S+T)$
3. $+A-B \times (C+D \uparrow (E-F))$

Having worked through these examples you will realize that the apparently rather complicated rules of section 3.3.1 essentially are only a concise formulation of the ordinary rules for writing arithmetic expressions.

(Time min.)

11 Problem 2. Some of the following sequences are arithmetic expressions, some are not. Mark those which are.

1. $a \times b/c \uparrow d/e \times f$
2. $+a \times -b$
3. $2_{10}6 \times 4.3 + Q$
4. $2 \times_{10}6/4.3$
5. $3.84_{10}(7+n)/4$
6. $PQ \uparrow +7.3$
7. $-(+(-v))$
8. $p/qrs \times tu-v$

(Time min.)

11 Problem 3. For each of the correct arithmetic expressions of 11 Problem 2 write a reasonable type declaration for the variables which occur in the expression. (Time min.)

11 Problem 4. Assuming that at a certain point in a program the values of seven simple variables are as follows:

va = 2, vb = 3, vc = 4, vd = 5, ve = 6, vf = 7, vg = 8,

find the values of the following expressions:

1. $va + vc \times vb / ve$
2. $vd \times (vc + vg)/ ve / va$
3. $vc \uparrow (vd - vb)$
4. $vf \uparrow va \times (vf - vc) / vb / (vb + vc)$
5. $va \times (vb \times (vg - vb \uparrow va /(ve / va)) - 2 \times vd) / (vg - vb)$
6. $vc \uparrow vb \uparrow va$
7. $(vc \uparrow vb) \uparrow va$
8. $vc \uparrow (vb \uparrow va)$
9. $((((vb \times va - vc) \times (-va) + vd) \times va + ve) \times (vf - vd)) -vg + 2$
10. $vc \div (vg - vb)$
11. $(vg - vd) \uparrow vb \div ve$
12. $(ve - vf - va) \div vc$

(Time min.)

11 Problem 5. Write the following mathematical expressions as ALGOL expressions, without using redundant parentheses:

1. $S + \frac{s - t}{v^2}$

2. $(U - W)(1 - \frac{a^3}{k(a - k)})$

3. a^{n+m}

4. a^{b^n}

5. a^{b+s^n}

6. $(q^v)^g$

7. $\frac{p^q}{r^{s+t}}$

8. $\dfrac{a - \dfrac{b}{c(d - e^{f+q})}}{h^{i(j-k)} + q^{(\frac{m}{n+p})}}$

(Time min.)

12. Read sections 4.2.1 - 4.2.4 ignoring for the moment the references to subscripted variables and the entier function.

12 Example 1. As explained in greater detail later statements and declarations are normally separated by a semicolon and consecutive statements will normally be executed in the order in which they are written. Thus a part of a program might look like this:

```
real a, b, p, q ;
a := b := 7 ;
p := a + 3 × b - 2.3₁₀-1 ;
q := p + (a + 3)/(-b - 13) ;
a := p := q - b × 0.2 ;
```

In order to follow the action of these statements it is useful to write a table with a column for each variable, where each new value of this variable is entered. Such a table is shown below, where in addition the inserted numbers from 1 to 6 show the order in which the new values are formed.

a	b	p	q
1: 7	2: 7	3: 27.77	4: 27.27
5: 25.87		6: 25.87	

Thus the final values of a, b, p, and q, are 25.87, 7, 25.87, and 27.27, respectively. (Time min.)

12 Problem 1. Using the same system as in 12 Example 1, follow the action of the following statements and find the final values of the variables.

```
real r1, ra, rb ;
integer n, i, j ;
n := 5 ;
r1 := n/(n + 15) ;
rb := n + 6/(6 × r1 + 0.5) ;
i := n := n - 2 ;
j := rb - i ;
ra := (j - i) × r1 × (rb - 4) ;
r1 := ra + rb + n + i + j + 8 × r1 ;
rb := (r1 - rb × n + j - ra) ↑ (rb - j) + ra ;
j := n := 1 + n ÷ (j - 2) ;
i := n + ra ;
```

(Time min.)

13. Read sections 3.4.1 - 3.4.6.2. (Time min.)

13 Note 1. In DASK ALGOL the implication operator ⊃ is not included.

13 Problem 1. Using the same technique as the one explained in 11 Example 1, analyze the following Boolean expressions and find the number of relations, Boolean primaries, Boolean secondaries, Boolean factors, Boolean terms, implications, simple Booleans and Boolean expressions entering into each of them:

1. ¬((c≡s) ∧ (P > Q ∨ W))
2. u ↑ 2 > 17.2 ∨ W ∧ Q ∨ ¬ T

(Time min.)

13 Problem 2. For each of the expressions of 13 Problem 1, write suitable type declarations for the identifiers. (Time min.)

13 Problem 3. Using the same scheme as in 12 Example 1, work through the following statements and find the final values of all variables.

```
real ra, rb ;
integer ia ;
boolean ba, bb ;
ra := 7.5 ;
ia := 5 ;
rb := 3 × ra - 2 × ia ;
ba := rb > ia ∧ ia > ra ;
ra := 2 × (ra - ia) - 1 ;
ba := ¬ ra > ia ∨ ba ;
bb := (ba ≡ rb > ia) ∧ ra < rb ;
ba := ¬(ba ∨ bb) ;
```

(Time min.)

14. Convince yourself that according to section 3.5.1 a label may be an unsigned integer or an identifier and that a designational expression may be a label. Read sections 3.5.5 and 4 and the first three lines of section 4.1.1. Read sections 4.3.1 - 4.3.3.

14 Note 1. In DASK ALGOL unsigned integers cannot be used as labels.

14 Problem 1. The following statements generate a sequence of values for SUM. Find the first four of these values.

```
     real p, q, SUM ;
     integer n ;
     n := 1 ;
     p := 0.5 ;
     SUM := 0 ;
     q := 1 ;
loop:SUM := SUM + q/n ;
     q := q × p ;
     n := n + 1 ;
     go to loop ;                                      (Time     min.)
```

15. Read sections 4.4.1 - 4.4.3. (Time min.)

16. Read sections 4.5.1 - 4.5.4 ignoring for the moment those syntactic units which have not yet been defined during the course.

16 Note 1. The necessity of introducing the <unconditional statement> arises because a construction like

```
     if B1 then if B2 then S := exp else V := Q + 1 ;
```

must be avoided since its meaning is not clear. (Time min.)

16 Note 2. The basic point of the syntax of conditional statement is the following:

An if can never follow a then.

16 Problem 1. Using the system of 12 Example 1 follow the action of the following statements and find the final values of all variables.

```
         real u, W ;
         boolean B ;
         u := 3 ;
         B := true ;
repeat : W := u - 2 ;
         if u↑2 - 1/u > 0 ∧ W > -2 then u := 1/u
         else if B then go to Z
         else go to end ;
Z:       B := false ;
         u := W + 2 × u ;
         go to repeat ;
end:     B := u ≥ W                                   (Time     min.)
```

17. Read sections 4.1.1 - 4.1.3 ignoring the syntactic units which have not yet been covered: procedure statements, for statements. Read also section 5.

17 Note 1. Section 4.1.1 gives the important rules of how to join statements and declarations together to form a program. The main difficulty of this section is that of punctuation, particularly of when to write semicolon and when not to. The difficulty is directly connected with the use of the delimiter end. As a guide for the student the relevant rules may be restated as follows:

PUNCTUATION RULE 1: Within a program the first symbol following any statement (whether basic or not) must be one of the following three:

; else end

PUNCTUATION RULE 2: Any sequence . . . end end end . . . within a program must always be terminated by semicolon or else.

Punctuation rule 1 follows directly from the syntactic rules governing statements (sections 4.1.1, 4.5.1, 4.6.1, and 5.4.1). Punctuation rule 2 follows from observing that an end, whenever it occurs, is the last symbol of some statement, and then applying punctuation rule 1.

(Time min.)

17 Note 2. Recall the special comment conventions for end (section 2.3).

17 Note 3. In DASK ALGOL the declarations in a block head cannot be given in an arbitrary order, but must appear in the following order:

First: type declarations
Second: array -
Third: switch -
Fourth: procedure -

17 Example 1. The concept local may be illustrated by an example of a program structure as follows:

```
L1:   begin real A, B, C ;
            . . . . . .
L2:   P:    A := B + 2 × C ;
            . . . . . .
L3:         begin real A, D ;
                . . . . . .
L4:         Q:  A := 2 × B + C ;
L5:             D := 2 + B + A ;
                . . . . .
L6:         P:  C := 2 × A - D ;
                . . . . . .
L7:             go to P ;
                . . . . . .
L8:             go to R ;
                . . . . . .
L9:         end ;
            . . . . . .
L10:  R:    go to P ;
            . . . . .
L11:  end
```

Here we have a larger block, from L1 to L11, containing as one statement a smaller block from L3 to L9. In the outer block we work with the identifiers A, B, and C, which are local to this block. In the statement at L2 a value is assigned to this A. The inner block introduces a new, local, A and a D. This A, then, has no relation to the A of the outer block, which is now screened. The variables B and C, on the other hand, are the same in both blocks. At L4 they are used to assign a value to the local A. This value is again used to assign a value to the local D at L5 . These operations make no use whatsoever of the A of the outer block. At L6 a value is assigned to the non-local C, using the local A and D. Labels are automatically local. Thus the labels Q and P at L4 and L6 are only accessible from inside the inner block. The go to statement at L7 will therefore lead to the statement at L6. The go to statement at L8, on the other hand, will lead out of the inner block to L10 because the identifier R, being not declared in the inner block, will be non-local. The moment this passage out of the inner block occurs the local variables A and D are completely lost. The go to statement at L10 will lead to L2 because the label P at L6 is local to the inner block and thus inaccessible from L10. (Time min.)

17 Problem 1. Using the system of 12 Example 1 follow the action of the following program and find the values of those variables which are defined at the label STOP.

```
        begin real W, S, B, C ;
L1:      W := 8 ;
L2:      S := 3 ;
L3:      B := 2 × W - S ;
L4:      C := B - W ;
        begin real P, W ;
L5:           W := B - 2 × C ;
L6:           P := C↑2 - B ;
L7:     AA:   W := P - 2 × W ;
L8:           C := C + 1 ;
L9:           if W > 1 then go to AA ;
L10:          S := W - P + S
        end ;
L11:    W := W - C + S ;
        STOP:
        end ;
```

(Time min.)

17 Problem 2. Check the syntactic structure of the program of 17 Problem 1 against the rules of section 4.1.1 and find the number of unlabelled basic statements, basic statements, unconditional statements, statements, compound tails, block heads, unlabelled compounds, unlabelled blocks, compound statements, and blocks. (Time min.)

18. Read section 2.7.

18 Note 1. The scope of a label comprises, so to speak, all those statements from which the label may be seen.

18 Note 2. The definition of scope should be changed to read:

The scope of a quantity is the set of basic statements, if clauses and for clauses . . . .

18 Example 1. The concept of scope may be illustrated by the example given in 17 Example 1. The scopes of the different quantities are as follows:

	Scope includes statements at
A and P in outer block	L2 L10
B, C, and R	L2, L4, L5, L6, L7, L8, L10
D, Q, and A and P in inner block	L4, L5, L6, L7, L8

(Time min.)

18 Problem 1. Find the scopes of all the identifiers of 17 Problem 1. (Time min.)

18 Note 3. The meaning of the second paragraph of section 2.4.3 should now be clear.

19. Read sections 5.2.1 - 5.2.4.4. (Time min.)

19 Note 1. In DASK ALGOL <u>own</u> <u>arrays</u> cannot be handled (cf. MANUAL section 7.12).

19 Problem 1. Write a declaration for the following arrays:

MatA and MatB, having two subscripts, the first running from 1 to k, the second from 1 to n,

Zoop, having four subscripts, the first running from -7 to +7, the second from 1 to 10, and the third and fourth from 0 to 1. (Time min.)

20. Read (revise) sections 3.1.1 - 3.1.4.2, paying special attention to the subscripted variables. (Time min.)

21. Convince yourself that according to section 3.3.1 subscripted variables may be used in the same way as simple variables in arithmetic expressions. (Time min.)

22. Read (revise) sections 4.2.1 - 4.2.4. (Time min.)

22 Note 1. In the fourth example of section 4.2.2 there is a mistake in some of the editions of the ALGOL 60 report. The first symbol should be S (not s, cf. section 2.4.3). (Time min.)

22 Example 1. The detailed explanations of sections 4.2.3.1 - 4.2.3.3 are relevant in a case like:

```
    real n ; array A[1 : 10] ;
    n := 2 ;
    A[n + 1] := n := n + 2 ;
```

Section 4.2.3.1 produces:

```
    A[3] := n :=
```

Section 4.2.3.2 gives the value of the expression as 4.

Section 4.2.3.3 assigns 4 to n and A[3]. (Time min.)

22 Problem 1. Using again the system of 12 Example 1 follow the action of the following program and find the values of all variables at the label STOP.

```
begin integer i, j ; integer array A[1:3, 1:2], C[0:2] ;
j := i := 1 ;
C[j-1] := A[j,i] := j := i + 2 × j + 2 ;
A[2×i, C[j-2-3×i] - 3] := j - 2 × i ;
C[A[2,2×j-8]-3] := i := i + j ;
A[C[j-i+1]/2, 4×A[1,1] - 3×i] := A[1, 2×(i-j)] := A[2,2] - A[1,1] ;
i := - A[3,2] ;
j := i - j ;
A[i, -j-2] := C[i-1] := 7 ;
A[A[2,2], C[1] - C[0]] := C[i] := 2 × i ;
STOP:
end
```

(Time min.)

23. Read sections 3.2.1 - 3.2.5. Ignore the concepts <string> and <switch identifier> and the references to procedure declarations and procedure statements. If necessary use the alphabetic index at the end of the ALGOL 60 report to find the definitions of any other syntactic units.

24. Convince yourself that, according to section 3.3.1, function designators may be used in arithmetic expressions in the same way as simple variables. (Time min.)

24 Problem 1. Follow the action of the following statements and find the final values of all variables.

```
begin real r, p, s, log ;
p := 4 × arctan(1) ;
r := 4 × sin(p/6) ;
p := p/r ;
s := 5 + cos(p × sqrt(2×r↑2 + 1)) ;
r := sign(r↑3 - 2 × s)×(s - r) ;
log := ln(s×(s+r))/ln(10) ;
p := p × (s+r)
end
```

(Time min.)

24 Problem 2. Write an algorithm for calculating the complete solution of the second order equation

$$Az^2 + Bz + C = 0$$

The algorithm should be written as a block having the real variables A, B, and C, supplied from outside and itself supplying the solution in the form of two complex numbers. These should be expressed as four real variables using the following identifiers:

z1r real part of first solution
z1i imaginary part of first solution
z2r real part of second solution
z2i imaginary part of second solution.

The quantities which have a meaning outside the block of course should not be declared in the block head.

The solutions are given by the usual formula:

$$z = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

If B^2-4AC is negative this formula should be used for finding both of the complex solutions. If, however, B^2-4AC is positive the following method should be used for avoiding forming the numerator as the difference between two nearly equal numbers: The above formula should be used only for finding one of the roots, namely the one which results from taking that sign of the square root which makes the numerator be formed as the sum of two numbers of equal sign in other words from taking + the square root when B is negative and - the square root when B is positive. The other real root may then be formed from

$$z2 = \frac{C}{A \times z1}$$

where z1 denotes the first root.

If A=0 the equation is linear and should be solved as such. If also B=0 the algorithm should go to a label outside the block called INDETERMINATE.

If the two solutions degenerate to one both z1 and z2 should be set equal to the correct solution. If the solutions are real the imaginary parts should, of course, be set to zero.

Check your algorithm by following the action of it for the following sets of the parameters:

Parameter set	A	B	C
1	0	0	2
2	0	4	8
3	2	0	-8
4	1	-10	9
5	-1	+10	-9
6	-1	-4	-4
7	2	-8	26
8	4	0	0

(Time min.)

25. Read (revise) sections 3.3.1 - 3.3.3 paying special attention to the mechanism of the if clause and <u>else</u> (see particularly the second paragraph of section 3.3.3).

25 Note 1. In an expression like

```
if B then p else q + r
```

it is important to notice that the meaning is equivalent to

```
if B then p else (q + r)
```

and not equivalent to

```
(if B then p else q) + r
```

The reason for this is the following: The + in the original expression must, according to section 3.3.1, stand between a <simple arithmetic expression> on the left, and a <term> on the right. The <term> obviously is r. The <simple arithmetic expression> must be q. It cannot be

```
if B then p else q
```

since this is not a <simple arithmetic expression>. (Time min.)

25 Problem 1. Follow the action of the following statements and find the final values of all variables.

```
begin real a, b ;
     a := 7 ;
V:   b := if a > 10 then 15+a else 13-a ;
     a := 17 - b ;
     if a>b then go_to V ;
STOP:
end
```

(Time min.)

25 Problem 2. Find out whether the following construction is correct or not, and prove your conclusion on the basis of section 3.3.1:

```
A + if q<0 then 7 else 4
```

(Time min.)

25 Problem 3. Write an algorithm for finding the polar coordinates r and v when the rectangular coordinates x and y are given. This is equivalent to solving the equations

$$r \cos v = x$$
$$r \sin v = y$$

The angle v, which should lie in the range from 0 to 2π, should be determined through the use of the standard function arctan. The quadrant must, however, be determined from the sign of x or y. Be sure that your algorithm will work also for x and/or y = 0. If both are zero v should be set to zero.

Check your algorithm by following its action when x and y are given initially as follows:

Case:	1	2	3	4	5	6	7	8	9
x	0	1	1	0	-1	-1	-1	0	1
y	0	0	1	1	1	0	-1	-1	-1

(Time min.)

26. Read section 3.3.6 and MANUAL section 7.6. on the arithmetics. (Time min.)

27. Read (revise) section 3.4.1 paying special attention to the if clause and else. (Time min.)

27 Problem 1. Find the value of the sixth expression of section 3.4.2:

```
        if k < 1 then s > w else h ≤ c
```

for the following three sets of values of the variables:

	k	s	w	h	c
Set 1	-1	2	2	4	3
2	2	2	2	4	3
3	1	4	5	2	2

(Time min.)

27 Problem 2. Find the value of the last expression of section 3.4.2:

```
     if if if a then b else c then d else f then g else h<k
```

for the following three sets of values of the entering variables:

	a	b	c	d	f	g	h	k
Set 1	true	true	true	false	false	false	5	7
2	false	true	false	false	true	false	5	4
3	false	false	false	true	false	false	5	4

(Time min.)

28. Read sections 3.5.1 - 3.5.4 and 5.3.1 - 5.3.5. These sections are intimately bound together and cannot be understood without reference to each other. Read (revise) sections 4.3.1 - 4.3.5.

28 Note 1. The kind of situation referred to by the remark of section 5.3.5 may be illustrated by the following example:

```
begin switch W := tt, Q[n + 2] ;
      switch Q := Q1, Q2, Q3 ;
      . . . . .
      A:   begin real n ;
                 . . . . .
           TT:   go to W[2] ;
                 . . . . .
           end block A
end
```

The go to statement at TT refers to W[2]. The designational expression for W[2] is Q[n+2]. Into this expression the variable n enters. Owing to the declaration real n in the head of block A the statement TT is outside the scope of the n of Q[n+2]. Consequently the go to statement is undefined.

(Time min.)

28 Problem 1. Follow the action of the following statements, write a list of the labels to which the go to statements successively refer and find the final values of the variables:

```
begin integer n, s ;
switch S := SB, S2, S3, STOP ;
switch W := TW, S[n - s + 7] ;
       n := 7 ;
TW:    go to S[n - 4] ;
SB:    n := n - 1 ;
       s := s + n ;
       go to W[n - 2] ;
S3:    n := n - 2 ;
       s := n - 2 ;
       go to W[n - s - 1] ;
STOP:
end
```

(Time min.)

29. Read sections 4.6.1 - 4.6.6. (Time min.)

29 Note 1. The definition of <for statement> contains an ambiguity which has not yet been officially resolved. Until this happens it is recommended that it be corrected to read:

```
    <for statement>:= <for clause><unconditional statement>|
                      <label>:<for statement>
```

29 Note 2. In DASK ALGOL the controlled variable of a for clause can only be a simple variable, not a subscripted variable.

29 Problem 1. Find the values assigned to the controlled variable in the following for statements and the final value of s:

```
begin real p, q, r, s ; integer k, m ;
p := 1 ;   q := 2 ;   r := 3 ;   s := 0 ;
for k := p + q, q - p, r×p - q do s := s + k ;
for m := q step r until 7×q + 1 do s := s - m ;
for k := 2, s, 2 step 2 until 6 do s := s + 2×k ;
for m := s + 45 , m + 2 while s<0 do s := s - m ;
for k := 1 step 1 until 5 do
       for m := 3 step -1 until 0 do s := s + k + m ;
```

(Time min.)

29 Example 1. For statements are particularly useful for executing operations on vectors and matrices (described in ALGOL as arrays). A simple example is the addition of two vectors VA and VB to give a third VC. This may be expressed as

```
    integer i ; array VA, VB, VC [1 : n] ;
    for i := 1 step 1 until n do VC[i] := VA[i] + VB[i] ;
```

Note that the quantity n cannot be declared in the same block head as the arrays VA, VB, VC (cf. section 5.2.4.2).

29 Problem 2. Write a block for multiplying matrix A (subscripts from 1 to i and 1 to j) by matrix B (1 to j by 1 to k) to form a matrix C (1 to i by 1 to k). Mathematically this is expressed as

$$C_{pq} = \sum_s A_{ps} \times B_{sq}$$

(Time min.)

30. As an introduction to the study of the remaining part of the language, the procedure mechanism, the following notes may be of help.

The procedure concept essentially has developed from the desire of being able to introduce any needed extension the basic mechanisms of the language. A few examples of such extensions are matrix arithmetics, transcendental function such as Bessel functions, and integration of differential equations.

In ALGOL all such mechanisms may be expressed by means of procedures. The ALGOL procedure concept is based on procedure declarations and procedure statements. A procedure declaration is the means of defining a new mechanism and associating an identifier with it. Thus the essential part of a procedure declaration is a piece of more elementary ALGOL language, the so-called procedure body. The rest of the procedure declaration, the procedure heading, only serves to specify the manner in which the procedure body is connected with the rest of the program.

The procedure declaration never executes any operations by itself. In order to put the process defined in it to work it is necessary to call it by means of a procedure statement. This, then, may be thought of as a short hand description of the complete process defined in the procedure declaration. This is all the more apt since the same procedure may be called from any number of different places within the same program.

Now read section 5.4.1 - 5.4.6. If necessary use the alphabetic index of definitions. (Time min.)

30 Note 1. In agreement with the correction of 29 Note 1 the declaration for Absmax should be corrected as follows: Insert a <u>begin</u> immidiately before <u>if</u> and an <u>end</u> between the two <u>end</u>'s.

30 Problem 1. In each of the 5 examples of section 5.4.2 localize the procedure heading and its constituents: procedure identifier, formal parameter part, ; , value part, specification part, and also the parameter delimiters. Find the formal parameters. Finally for each of the identifiers in the procedure bodies find out whether it is local, formal, or non-local. (Time min.)

30 Problem 2. Quote the procedure identifier of those of the procedure declarations of section 5.4.2 which define the value of a function designator. (Time min.)

31. Read sections 4.7.1 - 4.7.4 and 4.7.7.

31 Example 1. The important rules of section 4.7.3 may be illustrated by the following elaboration of the examples of sections 4.7.2 and 5.4.2. The first procedure statement of section 4.7.2:

```
Spur(A)Order:(7)Result to:(V)
```

can only make sense if it occurs in a block where, besides the declaration for the procedure Spur, declarations for A and V hold as follows:

```
array A[1:7, 1:7] ; real V ;
```

Now the effect of the rule of section 4.7.3.1 will be to add the assignment statement

```
n := 7
```

and the declaration <u>integer</u> n at the head of the procedure body.

The effect of the rule of section 4.7.3.2 will be to replace a by A and s by V throughout the procedure body. Thus, the effect of the above procedure statement is the same as that of the following block

```
begin integer k, n ;
      n := 7 ;
      V := 0 ;
      for k := 1 step 1 until n do V := V + A[k,k]
end
```

(Time min.)

31 Problem 1. In the same way as in 31 Example 1 execute the operations of section 4.7.3 to find the effects of the remaining procedure statements of section 4.7.2:

```
Transpose (W, v+1)
Absmax (A, N, M, Yy, I, K)
Innerproduct (A[t, P, u], B[P], 10, P, Y)
```

(Time min.)

31 Problem 2. Assuming that the value part: <u>value</u> n were removed from the heading of the declaration of Transpose (section 5.4.2), what would be the effect of the procedure statement

```
Transpose (W, v+1)
```

(Time min.)

31 Problem 3. Find the values of the quantities R, I, and K, at the label FINIS of the following program (the declaration for Absmax is that of section 5.4.2):

```
begin array zero[1:2, 1:2] ; real R ; integer I, K ;
zero[1,1] := zero[1,2] := zero [2,1] := zero [2,2] := 0 ;
Absmax (zero)size:(2,2)Result:(R, I, K) ;
FINIS:
end ;
```

If you find the result unsatisfactory what improvement of the procedure declaration could you suggest. (Time min.)

32. Read sections 4.7.5 - 4.7.6.

32 Note 1. In DASK ALGOL it will not be possible to call arrays by value.

32 Note 2. In DASK ALGOL procedures calling themselves, or using their own identifier within their bodies recursively, cannot be handled.

32 Note 3. The remark of section 4.7.6 is closely related to that of section 5.3.5 (see 28 Note 1).

32 Example 1. Formal parameters should generally be called by value when they represent pure input data to the procedure, in other words when in the procedure statement they may correctly correspond to expressions. The effect of calling a formal parameter by value is

a) To screen the corresponding actual parameter, i.e. to make sure that it is left unaltered by the procedure statement.

b) To economize the procedure call in the case that an expression, and not just a simple variable, is entered in the corresponding position.

c) To allow the use of the formal parameter as an internal working variable of the procedure body.

The following example will serve to bring out these points:

```
procedure EX(A, B) ; value A ; real A, B ;
begin integer k ;
A := A↑2 - sin(A × (A↑3 - 1)) ;
B := 0 ;
for k := 1 step 1 until 5 do B := B + A × (B + 1)/k ↑5
end
```

If this procedure is called only as follows:

```
EX(a,b)
```

value A may correctly be omitted. In this case the value of the variable a would, however, be changed by the procedure statement. If the procedure is called as follows:

```
EX(p+q, b)
```

value A is necessary, since if it were not present the meaningless construction

```
p + q := (p+q)↑2 - sin((p+q) × ((p+q)↑3 - 1))
```

would result from the application of the rules of section 4.7.3. In addition value A evidently achieves an economy in evaluating the first basic statement of the procedure body, since the sum p+q is only evaluated once.

It should be noted, however, that not all pure input data should be called by value. An example of this is presented by the formal parameters a and b of the procedure Innerproduct of section 5.4.2. Evidently, the whole meaning of this procedure depends on the possibility of not calling these parameters by value. (Time min.)

32 Problem 1. Write the declaration for a procedure for solving second order equations, using the principles of 24 Problem 2. (Time min.)

32 Problem 2. Write a declaration for a procedure for finding the polar coordinates from the rectangular ones (cf. 25 Problem 3). (Time min.)

32 Example 2. If a procedure has no formal parameter part it must work on non-local quantities of the procedure body. An example would be the following:

```
procedure R ; Q := sqrt(x↑2 + y↑2)
```

This procedure works on the three non-local parameters Q, x, and y. These must, of course, have a scope which includes the block in the heading of which the above declaration occurs.

Another variant is

```
real procedure R ; R := sqrt(x↑2 + y↑2)
```

This must, to be useful, be used in expressions, e.g.

```
S := p + q + R
```

This example will serve to warn the reader that an apparently simple addition may, in fact, imply a procedure call.

32 Example 3. The most intractable consequences of ALGOL will be realized if the above possibilities are combined. Thus the procedure

```
real procedure Sneaky(z) ; value z ; real z ;
begin Sneaky := z + (z - 2)↑2 ;
     W := z + 1
end Sneaky
```

will, when used in an expression such as

```
     P := Sneaky(v - 1) + 2
```

cause a change of the value of W behind the back of the user, so to speak. Furthermore this construction will cause the effect of

```
     Pip := Sneaky(k) × W
```

to be different from that of

```
     Pip := W × Sneaky(k)
```

Evidently such possibilities, if used must be handled with utmost caution. (Time min.)

33. Read section 2.6.1 - 2.6.3.

33 Note 1. DASK ALGOL uses the symbol ⊥ for space and effectively two different kinds of string quotes:

```
    ≮     ≯          for layouts
    ≮<    ≯          for other strings
```

Strings within strings cannot be used.

34. Read sections 5.4.6, 4.7.8 and MANUAL sections 8 - 8.7 on DASK ALGOL STANDARD OUTPUT PROCEDURES.

34 Note 1. Most of the complications of the syntax of section 8.3.1 arise from the following restrictions:

1. The neighbours of a space symbol ⊥ on either side must be n, d, or 0, and cannot be . or another ⊥

2. The sequence of letters d and digits 0 may start with a number of d's and must be followed by a number of 0's, but the two cannot be mixed.

(Time min.)

34 Problem 1. Show the printed results of the following statements:

```
begin real p, q ;
p := 9 ;
q := 2/p ;
tryk vr ;
tryk tekst (≮<p⊥=⊥≯) ;
tryk (≮d.d≯, p)
tryk (≮±dd.dddd≯, -p+q≯)
end ;
```

(Time min.)

34 Problem 2. Write four layouts which will produce the numbers in the following four columns

```
, 12.34       , ,-.973 24, ,+  17      , , 7 777        ⏨ 5,
,  0.027 43,  ,+.000 12, ,- 230       , ,  -628.3          ,
,555.6        , ,-.013 45, ,+  15⏨+ 6, ,     -1.538 ⏨-10,
                           ,-1300⏨-12, ,      0.222 2
```

35. Read MANUAL sections 9 - 9.7 on STANDARD INPUT PROCEDURES.

35. Problem 1. Find the exact output from the following program when supplied with the input symbols shown below:

```
begin integer u, v, w ;
real procedure Innerproduct (a, b, k, p) ; value k ;
integer k, p ; real a, b ;
begin real s ;
      s:= 0 ; for p:= 1 step 1 until k do s:= s + a × b ;
      Innerproduct := s
end Innerproduct ;
```

PROGRAM:

```
trykkopi (⟨</;⟩);
T: læs (u, v, w);
begin integer P, Q, R;
      real array A[1:u, 1:v], B[1:u, 1:w];
      læs(A, B);
  S:  læsstreng; if streng (⟨<Ar⟩) then go to T;
      læs(Q, R); trykvr; tryk(⟨-dddd⟩, Q, R);
      tryk(⟨-dddd.dd⟩, Innerproduct(A[P, Q], B[P, R], u, P));
      goto S;
end
end;
```

```
Sample input data /
Example of læs, læsstreng, streng.
   Q     R Sum(A[i,Q]×B[i,R])
;
Arrays: u = 3, v = 2, w = 4,
A:
.1,    .2,
.3,    .4,
.5,    .6,
B:
1,   2,   3,   4,
5,   6,   7,   8,
9,  10,  11,  12,
Q, R:   1,   3,
Q  R:   2,   2,
Arrays: u = 2, v = 3, w = 2,
A:
-.9,    -.8,    -.7,
-.6,    -.5,    -.4,
B:
6,   7,
8,   9,
Q, R:  3,   2,
```

3 Problem 1. Assignment statements, labels.
4 Problem 1. 3,4,6,7,10,12.
5 Problem 1. 3,4,5,7,11.
5 Problem 2. 1. a:=b+3 ; V: PW:=n ;
    2. begin if PQ=0 then go to W ;
    3. Q[n]:=n+7 end else go to WW
    4. tu:=vu/2 end end ;
6 Problem 1.
    2 keys: + - × / < = > ∨ ∧ , · ₁₀ : ; ( ) [ ]
    3 keys: ↑ ≤ ≠ ¬ := ⊥ ⊃
    4 keys: ≥ ≡ ⊂
7 Problem 1. 1,4,6,7,11,12.
8 Problem 1. 1. +729300000. 2. 9812. 3. 1000. 4. -.000001834.
    5. -.000001. 6. -4800.
8 Problem 2. 1. $17_{10}3$. 2. $_{10}3$. 3. $-134_{10}-5$.
8 Problem 3. 1,2,7,9.
9 Problem 1. The 3 first.
10 Problem 1. 1,3,5,7,10,12.
11 Problem 1.

Expression:	1	2	3
Primaries:	3	5	8
Factors:	3	5	8
Terms:	3	5	7
Simple arith. expr.:	3	3	6
Arithmetic expr.:	3	2	3

11 Problem 2. 1, 3, 4, 7, 8.
11 Problem 3. 1. integer a, b ; real c, d, e, f. 3. real Q. 4. - .
    7. real v. 8. real p, qrs, tu, v.
11 Problem 4. 1: 4, 2: 5, 3: 16, 4: 7, 5: 2, 6: 4096, 7: 4096, 8: 2 to the
    18th power, 9: 10, 10: 0 11: 4, 12: 0.
11 Problem 5. 1. S + (s - t)/v↑2
    2. (U - W)×(1 - a ↑ 3 / k /(a - k))
    3. a ↑(n + m)
    4. a ↑(b↑n)
    5. a ↑(b + s ↑ n)
    6. q ↑ v ↑ g
    7. p ↑ q / r ↑(s + t)
    8. (a-b/c/(d-e↑(f+q)))/(h↑i↑(j-k)+q↑(m/(n+p)))
12 Problem 1. r1 = 23, ra = 2, rb = 10, n = 2, i = 4, j = 2.
13 Problem 1.

Expression no.	1	2
Relations	1	1
Boolean primaries	7	4
Boolean secondaries	7	4
Boolean factors	7	4
Boolean terms	6	3
Implications	5	1
Simple Boolean expressions	5	1
Boolean expressions	4	1

13 Problem 2. 1. boolean c, s, W ; real P, Q
    2. real u ; boolean W, Q, T
13 Problem 3. ra = 4, rb = 12.5, ia = 5 ba = false, bb = true
14 Problem 1. SUM = 0, 1, 1.25 1.333333..
16 Problem 1. B = true, u = 13/15 W = -17/15.

17 Problem 1. W = -8, S = -9, B = 13, C = 7.
17 Problem 2. Unlabelled basic statements: 12, basic statements: 24, unconditional statements: 26, statements: 28, compound tails: 28, block heads: 2, unlabelled compounds: 0, unlabelled blocks: 2, compound statements: 0, blocks: 2.
18 Problem 1. S, B, C: all statements. W in outer block: 1, 2, 3, 4, 11, STOP. W in inner block, P, AA : 5, 6, 7, 8, 9, 10.
19 Problem 1. array MatA, MatB [1:k, 1:n], Zoop [-7:+7, 1:10, 0:1, 0:1]
22 Problem 1. i=2, j=-3, A[1,1]=5, A[1,2]=-2, A[2,1]=7, A[2,2]=3, A[3,1]=4, A[3,2]=-2, C[0]=6, C[1]=7, C[2]=4.
24 Problem 1. p=3.14159.., r=-3, s=5, log=1.
25 Problem 1. a = -9, b = 26.
25 Problem 2. Not correct. if q<0 then 7 else 4 is not a <term>.
27 Problem 1. 1. false. 2. false. 3. true.
27 Problem 2. 1. true. 2. false. 3. false.
28 Problem 1. n = 4, s=7. S3, TW, SB, STOP.
29 Problem 1. k = 3, 1, 1. s = 5.

```
    m = 2, 5, 8, 11, 14. s = -35.
    k = 2, -31, 2, 4, 6. s = -69.
    m = -24, -22, -20, -18. s = 15.
    k = 1, m = 3, 2, 1, 0. s = 25.
    k = 2, m = 3, 2, 1, 0. s = 39.
    k = 3, m = 3, 2, 1, 0. s = 57.
    k = 4, m = 3, 2, 1, 0. s = 79.
    k = 5, m = 3, 2, 1, 0. s =105.
```

29 Problem 2. The arrays must be declared in a block outside of the block in which the matrix multiplication is carried out.

```
begin array A[1:i, 1:j], B[1:j, 1:k], C[1:i, 1:k] ;
     . . . . .
     begin integer m, n, p ; real s ;
     for m := 1 step 1 until i do
     for n := 1 step 1 until k do
         begin s := 0 ;
               for p := 1 step 1 until j do s := s + A[m,p]×B[p,n] ;
               C[m,n] := s;
               comment For running time economy the simple variable s,
               and not C[m,n], is used during the summation;
         end for m og n
     end block ;
     . . . . .
end outer block
```

30 Problem 2. Step.
31 Problem 1. Transpose (W, v+1) will be executed as:

```
begin real w ; integer i, k, n ;
     n := v + 1 ;
     for i := 1 step 1 until n do
          for k := 1+i step 1 until n do
          begin w := W[i,k] ;
                W[i,k] := W[k,i] ;
                W[k,i] := w
          end for k
end
```

Absmax (A, N, M, Yy, I, K) will be executed as:

```
begin integer p, q ;
Yy := 0 ;
for p := 1 step 1 until N do for q := 1 step 1 until M do
begin if abs(A[p,q]) > Yy then
     begin Yy := abs(A[p,q]) ; I := p ; K := q end
end for p
end procedure Absmax
```

Note that an extra begin end bracket has been inserted in order to make the statement following do unconditional.

Innerproduct (A[t, P, u], B[P], 10, P, Y) will be executed as:

```
begin real s ; integer k ;
k := 10 ; s := 0 ;
for P := 1 step 1 until k do s := s + (A[t, P, u]) × (B[P]) ;
Y := s
end Innerproduct
```

31 Problem 2.

```
begin real w ; integer i, k ;
for i := 1 step 1 until (v+1) do
    for k := 1+i step 1 until (v+1) do
    begin w := W[i,k] ; W[i,k] := W[k,i] ; W[k,i] := w end
end Transpose
```

31 Problem 3. R = 0, I and K are undefined. Since the user must expect that all of these quantities are defined upon exit from the procedure this is unsatisfactory. Two possible improvements of the procedure declaration may be suggested to remedy this: 1. Replace the first statement of the procedure body by, e.g., y := -1. 2. Replace the relational operator > by ≥ .

32 Problem 1.

```
procedure EQ2OR (A, B, C, z1r, z1i, z2r, z2i, INDETERMINATE) ;
value A, B, C ; real A, B, C, z1r, z1i, z2r, z2i ; label INDETERMINATE ;
begin real discriminant ;
                if A ≠ 0 then go to normal ;
                if B = 0 then go to INDETERMINATE ;
                z1r := z2r := - C/B ; go to set zero ;
normal:         discriminant := B ↑ 2 - 4 × A × C ;
                if discriminant > 0 then go to real solution ;
complex:        z1r := z2r := - B / 2 / A ;
                z1i := sqrt(-discriminant)/2/A ;
                z2i := - z1i ; go to finis ;
real solution:  z1r := (-B+(if B>0 then -1 else 1)×sqrt(discriminant))/2/A;
                z2r := C/A/z1r ;
set zero:       z1i := z2i := 0 ;
finis:
end EQ2OR
```

32 Problem 2.

```
procedure Polar (x, y, r, v) ; value x, y ; real x, y, r, v ;
begin r := sqrt(x↑2 + y↑2) ;
      v := if y=0 then (if x≥0 then 0 else 3.14159265)
           else arctan(-x/y) + (if y>0 then 1.5707963 else 4.7123889)
end
```

34 Problem 1.
p = 9.0- 8.7778

34 Problem 2. ndd.d00⏨00 +.ddd dd ±dd00⏨±dd -n⏨ddd.000⏨0⏨-dd

35 Problem 1.
Example of læs, læsstreng, streng.

Q	R	Sum(A[i,Q]×B[i,R])
1	3	7.90
2	2	8.80
3	2	-8.50

A PROGRAM FOR A SMALL TABLE.

An illustration of ALGOL.

As an illustration of the use of ALGOL the complete solution of a simple problem is given below. The additional notes will enable the reader to pick up some of the basic features of the language in an informal manner.

It should be noted that the ALGOL program gives a complete description of the solution of the problem. Indeed, an ALGOL translator system will be able to build up a complete machine code for the solution on the basis of the ALGOL program in precisely the form given below. Both the translation and the solution will be performed with the speed and efficiency characteristic of the electronic calculators. Consequently, once the ALGOL program has been written the problem is practically solved. There remains only a purely routine operation of the electronic calculating machine.

Definition of the problem.

It is desired to calculate a table of the following function:

$$\text{Acab}(u,\ \text{length}) = \frac{u(\text{length}^2 - 0.037\ u^3)}{2u^2 + \underset{\text{length}}{\cancel{\text{langde}}}^4}$$

The parameter u varies from 0,0 to 5.0 in steps of 0.2. The parameter length assumes the following six values

length = 1.0, 1.2, 1.4, 1.6, 1.8, 2.0.

The results should be printed in a table with seven columns and a heading as shown below (the commas indicate spaces):

```
,,,,,,,,,,,,,,,,,,,,,,,,,,,Table of function Acab.
,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,length
,u,,,,,,,1.0,,,,,,,,1.2,,,,,,,,1.4,,,,,,,,1.6,,,,,,,,1.8,,,,,,,,2.0
,
0.0,,,,,xx.xxx,,,,,xx.xxx,,,,,xx.xxx,,,,,xx.xxx,,,,,xx.xxx,,,,,xx.xxx
0.2,,,,,xx.xxx,,,,,xx.xxx ,,,,xx.xxx,,, ,xx.xxx,,,,,xx.xxx,, ,,xx.xxx
etc.
```

ALGOL program:	Notes:
<u>begin</u>	Any program must be embraced within the statement bracket <u>begin</u> <u>end</u>.
<u>comment</u> Program for Acab;	Notes in plain language may easily be included.
<u>real</u> u, length;	This declares that the quantities denoted by u and length represent arbitrary <u>real</u> numbers. There is a considerable freedom in choice of designations for quantities in ALGOL.
tryktom (50) ;	This is the first active instruction of the program. It causes 50 empty rows of tape to be punched by the output punch. This will facilitate the handling of the tape of results. - The following instruction punches the heading.
trykteket ({<	

```
                         Table of function Acab.

                                     length
 u        1.0          1.2          1.4         1.6          1.8          2.0
```

}) ;	
<u>for</u> u := 0.0 <u>step</u> 0.2 <u>until</u> 5.01 <u>do</u>	This construction causes the following statement (from <u>begin</u> to <u>end</u>) to be executed repeatedly by with u = 0.0, 0.2, 0.4, etc.
<u>begin</u> trykvr ;	Output of a carriage return code (vr = vognretur). This includes line feed.
tryk({d.d}, u) ;	u is printed with one decimal.
<u>for</u> ~~langde~~ length := 1.0 <u>step</u> 0.2 <u>until</u> 2.01 <u>do</u>	Inside the larger repetitive process a smaller repetition is performed, for the calculation and printing of the six columns in each line.
<u>begin</u> trykml (5) ;	In front of each function value 5 spaces (<u>mellemrum</u>) are printed.
tryk ({dd.ddd}, u×(length↑2-0.037×u↑3)/(2×u↑2+~~langde~~ length↑4))	The function value is calculated and printed in a layout of two digits before and three after the decimal point. Arithmetic expressions must be written linearly. Spaces and carriage returns may, however, be inserted freely.
<u>end</u>	This ends the section controlled by the <u>for</u> ~~langde~~ length construction.
<u>end</u> ;	This ends the section controlled by the <u>for</u> u construction.
tryktom (50)	Output of a suitable piece of empty tape.
<u>end</u> ;	This is the end of the program.

THE SOLUTION OF A REALISTIC PROBLEM.

The following formulation of a problem is taken over directly from that presented by a physicist:

It is desired to tabulate the following expressions for l_2 and Aber:

$$l_2 = -r \frac{\mathrm{tg}\, 2e_2 + \frac{1}{c}}{1 - \frac{1}{c}\,\mathrm{tg}\, 2e_2 + (\mathrm{tg}\, 2e_2 + \frac{1}{c})\,\mathrm{tg}\, e_2}$$

where

$$c = \frac{r}{l_1} + \mathrm{tg}\, e_1$$

(for $e_2 = -45^\circ$, l_2 becomes $\frac{rc}{r + c}$)

$$\text{Aber} = - H\, r\, [c_1 + c_2]$$

where

$$H = \frac{1}{2} \frac{l_1^2\, l_2}{r^3} [1 + (\frac{r}{l_1} + \mathrm{tg}\, e_1)^2] \Big/ \sqrt{1 + (\frac{r}{l_2} + \mathrm{tg}\, e_2)^2}$$

$$c_1 = \frac{r^2}{l_1^2} \frac{\frac{r}{l_1} + 3\,\mathrm{tg}\, e_1}{[1 + (\frac{r}{l_1} + \mathrm{tg}\, e_1)^2]^{3/2}}$$

$$c_2 = \frac{r^2}{l_2^2} \frac{\frac{r}{l_2} + 3\,\mathrm{tg}\, e_2}{[1 + (\frac{r}{l_2} + \mathrm{tg}\, e_2)^2]^{3/2}}$$

The parameter values are the following:

l_1 is 50
e_1 assumes the values 0 to 50 degrees, in steps of 5 degrees
e_2 assumes the values -20 to -50 degrees, in steps of 5 degrees
r assumes the values 30 to 120 in steps of 5

The results should be tabulated in 11 tables, one for each of the values of e_1, the value of which should be printed at the head of the table. The arrangement of the tables should be as follows:

l1 = 50, e1 = 20

e2 =	-20	-25	-30	-35	-40	-45	-50
r	12 Aber	12 Aber	12 Aber	12 Aber etc.			

The results, which will be smaller than 1000, should be printed with one decimal.

SOLUTION 1.

```
begin comment This is a direct, but uneconomical program for 12 and Aber;
integer 11, e1, e2, r ;
real 12, c, c1, c2, Aber ;
real procedure tg(u) ; value u ; real u ;
begin real COS ;
      u := u/57.2957795 ; COS := cos(u) ;
      tg := if COS=0 then ₁₀20 else sin(u)/COS
end tg ; comment It is easy to see that this way of treating the singula-
rity of tg is correct in the present application ;

BEGIN OF PROGRAM:
tryktom (50) ; 11 := 50 ;
for e1 := 0 step 5 until 50 do
    begin tryktekst({<
11␣=␣50,␣e1␣=␣}) ; tryk({dd}, e1) ;
tryktekst ({<
␣␣␣␣e2␣=␣␣␣␣-20␣␣␣␣␣␣␣␣␣␣␣␣␣␣-25␣␣␣␣␣␣␣␣␣␣␣␣␣␣-30␣␣␣␣␣␣␣␣␣␣␣␣␣␣-35}) ;
tryktekst   ({<␣␣␣␣␣␣␣␣␣␣␣␣␣-40␣␣␣␣␣␣␣␣␣␣␣␣␣-45␣␣␣␣␣␣␣␣␣␣␣␣␣␣-50
}) ;
for r := 30 step 5 until 120 do
    begin tryk vr; tryk({ddd}, r) ;
          c := r/11 + tg(e1) ;
          c1 := (r/11+3×tg(e1))×r↑2/11↑2/(1+(r/11+tg(e1))↑2)
                /sqrt(1+(r/11+tg(e1))↑2) ;
          for e2 := -20 step -5 until -50 do
              begin 12 := -r×(tg(2×e2)+1/c)/
                          (1-tg(2×e2)/c+(tg(2×e2)+1/c)×tg(e2)) ;
                    tryk ({-dddddd.d}, 12) ;
                    c2 := (r/12+3×tg(e2))×r↑2/12↑2/(1+(r/12+tg(e2))↑2)
                          /sqrt(1+(r/12+tg(e2))↑2) ;
                    Aber := -11↑2×12/2/r↑2×(1+(r/11+tg(e1))↑2)
                            ×sqrt(1+(r/12+tg(e2))↑2)×(c1+c2) ;
                    tryk ({dddd.d}, Aber)
              end for e2
    end for r ;
    tryk sum
    end for e1 ;
tryk tom (50)
end program ;
```

This program may be improved considerably, particularly with respect to efficiency. Obviously many parts of the expressions will be evaluated over and over again with the same numbers. This may be avoided by rewriting the formulae so as to evaluate as much of an expression as possible as soon as the entering quantities have been assigned. Also the repeated evaluations of tg(e2) may be avoided by preparing a table of this quantity. Finally the denominators of the formulae for c1 and c2 may conveniently be evaluated through a procedure. These features have all been incorporated in the following version of the program.

SOLUTION 2.

```
begin comment Improved program for l2 and Aber;
integer l1, e1, r, Q, i ;
real l2, crec, M, m1, m2, tge1, tge1t3, Aber ;
array tane2, tan2e2, tant3 [1:7] ;
real procedure tg(u) ; value u ; real u ;
begin real COS ;
      u := u/57.2957795 ; COS := cos(u) ;
      tg := if COS=0 then ₁₀20 else sin(u)/COS
end tg ; comment It is easy to see that this way of treating the singula-
rity of tg is correct in the present application ;
real procedure HELP (y); comment This helps to calculate the denominators
or c1 and c2 ; value y ; real y ;
begin y:=1+y↑2; HELP := 1/y/sqrt(y) end HELP ;

BEGIN OF PROGRAM:
for i := 1 step 1 until 7 do
begin tane2[i] := tg(-5 × i - 15) ;
      tan2e2[i]:= tg(-10 × i - 30) ;
      tant3[i] := 3 × tane2[i]
end for i ;
tryk tom(50) ; l1 := 50 ; Q := 1250 ;
for e1 := 0 step 5 until 50 do
    begin trykteskt(⟨<
l1␣=␣50,␣e1␣=␣⟩) ; tryk(⟨dd⟩, e1) ;
trykteskt (⟨<
␣␣␣␣e2␣=␣␣␣␣-20␣␣␣␣␣␣␣␣␣␣␣␣␣-25␣␣␣␣␣␣␣␣␣␣␣␣␣-30␣␣␣␣␣␣␣␣␣␣␣␣␣-35⟩) ;
trykteskt   (⟨<␣␣␣␣␣␣␣␣␣␣␣␣␣-40␣␣␣␣␣␣␣␣␣␣␣␣␣-45␣␣␣␣␣␣␣␣␣␣␣␣␣-50
⟩) ;
tge1 := tg(e1) ; tge1t3 := 3 × tge1 ;
for r := 30 step 5 until 120 do
    begin tryk vr; tryk(⟨ddd⟩ ,r) ;
          crec := 1/(r/l1 + tge1) ;
          M := Q × ((r/l1 + tge1)↑2 + 1) ;
          m1 := (r/l1 + tge1t3)/l1↑2 × HELP(r/l1 + tge1) ;
          for i := 1 step 1 until 7 do
              begin l2 := -r × (tan2e2[i] + crec)
                        /(1-tan2e2[i]×crec+(tan2e2[i]+crec)×tane2[i]) ;
                    tryk (⟨-dddddd.d⟩, e2) ;
                    m2 := (r/l2+tant3[i])/l2↑2×HELP(r/l2+tane2[i]) ;
                    Aber := -M×l2×sqrt((r/l2+tane2[i])↑2+1)×(m1+m2) ;
                    tryk(⟨-dddd.d⟩, Aber)
              end for e2
    end for r ;
    tryk sum
    end for e1 ;
tryk tom (50)
end program ;
```

It will be clear from this example firstly that the efficiency by which a process will be carried out may be improved even by just a simple revision of the formulae. Secondly that the establishment of the most suitable formulae in a given case depends directly on the desired form of the output.

THE TESTING OF ALGORITHMS.

Experience shows that it is rare for an algorithm to be correct when it is first written up. The testing of algorithms must therefore be considered to be a very important part of ALGOL programming. The following notes are intended as a first guide to this subject.

Errors in an ALGOL program may be of two essentially different kinds: (1) errors of form and (2) errors of content. In testing an algorithm these two kinds of errors should be treated sepatately.

Errors of form (syntax).

Errors of form (syntactical errors) may be eliminated completely through a purely mechanical process. Indeed it is possible to let the ALGOL-to-machine-code translator perform syntactic checking and reject incorrect programs. Likewise a manual checking may be (and should be!) performed in a routine manner. In ALGOL programs this is a comparatively easy matter owing to the easily readable form of the language. In performing the check the following list of some frequent errors may be useful:

1. Forgotten or wrong occurrence of ; or **else** or **end** (cf. the punctuation rules 1 and 2 , point 17 Note 1).
2. Declarations of simple variables forgotten.
3. Multiplication symbol × omitted.
4. **then** omitted (there must be one for every **if**).
5. Underlining of basic symbols forgotten.
6. Mixture of **integer** and **real** type variables on the left side of assignment statements.

Errors of content.

Errors of content are errors which cause the algorithm to perform a different action from the one intended. Since the description of the intended action is often vague and leaves a considerable freedom for the writer of the algorithm the detection of this type of error may often be quite difficult. Even so there are some general suggestions which may be of help:

1. For each variable check that it is never used before a value has been assigned to it.
2. Make sure that no division by zero or any other undefined operation (ln, sqrt, etc.) can occur.
3. Check for special values of input parameters, particularly zero.
4. Remember to take absolute value when doing test on magnitudes of quantities.
5. For each if clause of the program establish two test situations one which makes the Boolean **true** and one which makes it **false**, and check that the algorithm behaves correctly in both cases by following its action statement by statement.
6. Note that the method of following an algorithm step by step, as explained in point 12 Example 1, far from being a beginners device must be considered as the basic method for testing algorithms. When combined with a choice of values of input parameters made according to points 3 and 5 above it is the most efficient method for constructing correct algorithms.

THE USE OF BLOCKS AND PROCEDURES.

An important step in the planning of an ALGOL program is the subdivision of the process into parts which may conveniently be written as blocks or procedures. In order to be able to do this the programmer must have a clear idea of the properties of these ALGOL units. As a first introduction the following notes may be useful.

Blocks are useful for expressing such parts of the program which form a closed process. In particular a block is indispensable if in a process an array is needed whose size depends on the results of previous calculations. Such an array must be local to a block. In addition any other quantity (simple variable, label, switch, procedure) which is used only internally during the work of the block, but which has no interest when this work is done may be declared to be local to the block. This is particularly useful when different blocks of a program are written by different programmers. By using blocks the programmers will only have to agree on the non-local identifiers of the blocks, while inside each block the programmer is free to choose the identifiers of working quantities.

Procedures have three different important uses: 1. Generalization of the use of blocks. 2. Abbreviation of small ad-hoc functions. 3. Form of communication of closed processes between programmers at different times and places.

1. Any block may be converted into a procedure by adding a heading to it. The heading will attach an identifier to the block and usually name some or all of the non-local identifiers as formal. Where the block in question is written specially for the program this conversion can be recommended only if the mechanism of the block is used two or more times with different non-local quantities, corresponding to two or more calls of the procedure, since evidently a call of a procedure is a more elaborate process than a simple entrance into the corresponding block.

2. Frequently the formulae of a program may be shortened through the use of suitable function designators. As in 1 above this will be economical only if the corresponding ad-hoc procedure is used more than once during the program.

3. In a near future it is safe to expect that all important methods of numerical analysis will be expressed in the form of ALGOL procedures and published (cf. the Algorithms section of the Comm. ACM and the ALGOL Programming section of BIT). Since these procedures presumably will be above average with respect to efficiency it is strongly recommended that they be used wherever possible.