STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

REKENAFDELING

MR 64

User's Directions for the Second MC ALGOL 60 Translator

by

P. J. J. van de Laarschot

and

J. Nederkoorn

August, 1963

Table of Contents

This is a direction for the use of the second MC ALGOL translator. It contains only the matter that is absolutely indispensable to the user. It is the first part of a series of reports that will describe completely the second MC ALGOL translator.

The second translator differs from the first one in the following respects:

1. The elaboration aims at perfection; still more rigorous than in the first translator, speed has been sacrificed in behalf of completeness, shortness and simplicity in use.

2. The text offered for translation is checked thoroughly against the rules of ALGOL 60.

3. Own dynamic arrays are admissible now.

4. There is no restriction as to the length of identifiers.

5. An element of a Boolean array is stored in just one bit of a memory word; thus, a word can contain 27 Boolean array elements.

6. The program is executed interpretatively and a computer with variable word length is simulated.

7. Up to now, there is one complex, named ALP, that stores a real number in one memory word.

The complete system, consisting of three programs:

a. the translator
b. the program "Input of Object Program"
c. the complex ALP

was designed under the supervision of (to date: Professor) dr. E.W. Dijkstra and programmed in X1 machine code by the authors of this report. It is practically useful mainly as a text tester (viz. 2 hereabove) and for the execution of programs that will exceed the memory capacity of the installation if executed by means of the first translator.

Introduction

Good calculators will not use counting rods.
Lao Tse

Necessary apparatus:

1. an X1 computer fitted out with at least two memory cabinets, a tape reader and a tape punch.
2. a special type off-line Flexowriter (off-line = not connected with the X1).

The work to be done by the user consists of three distinct parts per program:

a. writing the program in ALGOL 60, preparation of a list or lists of input data and punching both on tapes;
b. translating the program (where necessary after testing and correcting the tape);
c. execution of the program once or more often, making use of various data input tapes for every run if so desired.

The ALGOL program and the data input tapes are punched by means of the Flexowriter.

The user of the system who has not two memory cabinets in the X1 installation at his disposal can have his program translated on some other X1 installation and perform the execution of the program on his own one.

For the matter of understanding this report acquaintance with the X1 is dispensable and one does not need any training in machine code programming. Operating the X1 for the execution of any ALGOL program translated with this system can be learned at the computer itself in a few minutes.

For a. hereabove see [1] and the chapters 1 through 4 of this report; for b. and c. see chapter 5. [2] may be used as an introduction into the matter, but the restrictions thereby imposed on the use of certain features admissible in pure ALGOL 60 are not, generally speaking, valid here.

Literature:

[1] Revised Report on the Algorithmic Language ALGOL 60, edited by Peter Naur, Regnecentralen, Copenhagen, 1962.
[2] A Primer of ALGOL 60 Programmming by Dr. E.W.Dijkstra, A.P.I.C. Studies in Data Processing, Nr 2, Ac.Press, London, 1962.
[3] Voorlopige Gebruiksaanwijzing van de Tweede MC ALGOL 60 Vertaler door P.J.J.van de Laarschot en J.Nederkoorn, MR 54, January 1963.
[4] Text of the Second ALGOL 60 Translator for the X1 by P.J.J.van de Laarschot en J.Nederkoorn, MR 61, May 1963.

Chapter 1

Writing an ALGOL program

The program must be a correct ALGOL 60 program as understood by [1]. With the exception of some punching indications the hardware representation is identical with the reference language.

We will now give - in the form of a sequential commentary on [1] - a list of restrictions, elucidations and interpretations of what we think are ambiguities.

(2.3) A space and transition to a new line (although not allowed in underlined and composite symbols such as begin and :=) are not elements of the language. However, in strings they denote themselves.

String quotes occurring in a comment should be nested in pairs.

(2.5.4) The absolute value of a variable which has been declared integer is less than 2 ↑ 26. A constant with a greater value occurring in the program is treated as if it were a real number. See chapter 3, 1st paragraph.

(3) The values of primaries occurring in an expression are calculated sequentially from left to right.

(3.2.4) Chapter 2 gives information about standard functions.

(3.5) Only identifiers can denote labels.

(4.1) A block must not be enclosed by more than 30 blocks. As this restriction is valid only in a statical sense one can elude it by procedure declaration on a lower level.

(4.6.1)
```
<for statement> ::= <for clause> <unconditional statement> |
                    <for clause> <for statement> |
                    <label> : <for statement>
```
This "do if - prohibition" can always be eluded by the insertion of statement brackets, e.g.

```
        for n:= 1, 2 do begin if x = 0 then S end
```

(4.6.3) A for statement is a block.

(4.7.3.1) The values of parameters quoted in the value part of the procedure declaration are calculated in the order in which these parameters have been specified. The value assignments are performed immediately after entering the procedure body.

The variables quoted in the value part are treated as if they were local to the procedure body, which means that an operation performed on them by the procedure cannot in any way have effect on the values which the corresponding actual parameters have in the main program.

(5) The validation of the declarations at the time of entry into a block is performed before the lower upper bound expressions in array declarations are evaluated.

Activating a procedure recursively is not considered as an exit from, or as an entry into a block as understood in the definition of <u>own</u>.

Upon exit from a block <u>own</u> values that belong to dynamically subordinated activations of that block become undefined.

(5.2.3.2) An array cannot have more than 31 dimensions.

(5.2.4.2) It is sufficient if the bounds have been defined. Thus, the expression "can only depend on" has been understood as liberally as possible.

(5.3.3) A switch list cannot contain more than 63 elements.

(5.4.1) A formal parameter list cannot contain more than 31 elements.

(5.4.3) The identifier of an array or variable must not be used before it has been declared. The natural exception to this rule is a formal parameter.

(5.4.5) Every formal parameter must be specified.

(5.4.6) A procedure written in non-ALGOL code is not a part of an ALGOL program. However, i n t h e c o m p l e x some of these non-ALGOL procedures have been incorporated and the possibility of adding more of them has been provided for. This means that machine code programs and standard functions are treated in an analoguous manner.

In chapter 2 we give a survey of the procedures in question.

Chapter 2

Machine Code Programs and Standard Functions

At most 104 machine code programs (MCP's) can be inserted into the complex. The MCP libraries of the first and the second ALGOL 60 translator of the Mathematical Centre up to now have been kept as identical as possible, if considered from the user's point of view.

Abbreviations:

E	arithmetical expression
IE	integer expression
F	function designator
NF	non - function procedure.

The user should keep in mind that in ALGOL 60 a function procedure can occur as a statement.

The procedures that will follow now can be used without a declaration. We give a very short description only, mostly in pseudo-ALGOL; for complete information refer to the loose-leaf edition of the Mathematical Centre, the series AP 100; the times mentioned there are, in general, not valid for the second ALGOL translator.

abs (E)	F	See (3.2.4) of [1]
sign (E)	F	"
sqrt (E)	F	"
sin (E)	F	"
cos (E)	F	"
arctan (E)	F	"
ln (E)	F	"
exp (E)	F	"
entier (E)	F	See (3.2.5) of [1].
TAB	NF	Directs the typewriter to the next tabulator stop in the same line.
NLCR	NF	Directs the typewriter to the first tabulator stop in the next line.
stop	NF	Stops the X1; manipulation of the key "Start Next Address" will then cause the X1 to resume the execution of the program, beginning with the statement following "stop".
SPACE (IE)	NF	Sends IE space instructions to the typewriter.
read	F	read:= next number of input tape. The type of read depends on the number offered.
FIXT (n, m, x)	NF	n and m are IE's, x is an E; n, m $\geq$ 0; prints the sign, n decimal digits of the integer part of x, a decimal point (unless m = 0), m decimal digits of the fractional part of x, space.

FLOT (n, x)	NF	n is an IE, x is an E. Computes $\underline{X}$ and d from $x = \underline{X} \times {}_{10}d$ and $.1 \leq abs(\underline{X}) < 1$; prints the sign of $\underline{X}$, point, n decimal digits of $\underline{X}$, space, sign of d, d in two decimal digits, space.
print (x)	NF	if type (x) = integer then begin FIXT (8, 0, x); TAB end else begin FLOT (8, x); TAB end (in this case FLOT prints d in three decimal digits)
PRINTTEXT (<string>)	NF	Prints the open string that is deduced from the parameter string by canceling the outermost string quotes. From the ALGOL symbols, only digits, letters, the symbols: . , ; : ( ) + - / and = and further space, tab and carriage return can be reproduced by the typewriter of the X1

SUM (i, h, k, ti)	F	i, h and k are IE's, ti is an E.
		begin real s; s:= 0;
		    for i:= h step 1 until k do s:= s + ti;
		    SUM:= s
		end

XEEN (IE)	F	XEEN:= in binary representation, the integer that has a "one" in every position where IE and the console word both have a "one", and zeros in all other positions. (In the console word, key up = 1, key down = 0)
EVEN (IE)	F	EVEN:= $(-1) \uparrow$ IE

INPROD (k, a, b, x, y)	F	k, a, b, x and y are E's. x and y are variables, with indices as a rule. The indices must be linear functions of k.
		begin real s; s:= 0;
		    for k:= a step 1 until b do
		    s:= s + x × y; INPROD:= s
		end

SETRANDOM (a)	NF	Necessary preparation for RANDOM. $0 \leq a < 1 - 2 \uparrow (-27)$
RANDOM	F	RANDOM:= next number out of a pseudo random sequence initiated with "a" (see SETRANDOM). $0 \leq$ RANDOM < 1

FACTOR (E)	F	FACTOR:= smallest factor greater than 1 of E. $\text{abs}\ (E) < 2 \uparrow 26$

REMAINDER (a, b)	F	a and b are E's. $\text{abs}\ (E) < 2 \uparrow 26$.
		REMAINDER:= if b = 0 then a
		else $a - a \mathbin{\underline{:}} b \times b$

GCD (a, b)	F	a and b are E's. $\text{abs}\ (E) < 2 \uparrow 26$.
		GCD:= if b = 0 then abs (a)
		else GCD (b, $a - a \mathbin{\underline{:}} b \times b$)

ABSFIXT (m, n, x)	NF	Like FIXT, but the sign is replaced by space.
RE7BIT	F	RE7BIT:= numerical value of the next heptade in input tape (so, $0 \leq \text{RE7BIT} < 128$).
PU7BIT (IE)	NF	Punches a heptade with numerical value = IE. $0 \leq \text{IE} < 128$

The MCP's that follow hereafter are designed to facilitate off-line output via tape punch and Flexowriter. They deliver punched tape that can be fed into the tape reader of the Flexowriter, which will then deliver a printed copy of the information in the tape. With these MCP's one can control the lay-out of that copy very much like one does when the output is printed by the typewriter of the X1.

The first call of some punching MCP in the course of a program will automatically generate a punching "carriage return". After that, a punching "carriage return" is given after 150 symbols at most in one line (here, of course, a space is counted as a symbol).

In a great majority of cases, punching is to be preferred to printing as it will save much valuable computer time.

PUTEXT (<string>)	NF	Is the punching analogon of PRINTTEXT. The parameter string, however, can contain all ALGOL symbols (underlined words excepted) and will appear between apostrophes on the printed output copy.
PUSPACE (IE)	NF	Punching analogon of SPACE.
PUNLCR	NF	Punching analogon of NLCR, but without the TAB contained therein.
RUNOUT	NF	Punches 40 cm of blank tape. This has not any effect on the printed copy.
TAPEND	NF	Punches an interrogation mark (to mark off the end of a sequence of numbers) followed by 40 cm of blank tape.

STOPCODE	NF	Punches a stop code, which is a signal for the Flexowriter to stop its activities, followed by 40 cm of blank tape.
FLOP (n, m, x)	NF	Punching analogon of FLOT (n, x), replacing the first space by "$_{10}$" and punching d in m decimal digits (m = 1, 2 or 3)
FIXP (n, m, x)	NF	Punching analogon of FIXT.
ABSFIXP (n, m, x)	NF	Punching analogon of ABSFIXT.

Chapter 3

Exactness, Computing Methods and Memory Space

Exactness and Computing Methods

An integer is memorized in one X1 memory word of 27 bits (= binary digits); its absolute value is less than $2 \uparrow 26 = 67\ 108864$. If, in the course of a computation, a greater integer number is generated, or if in the program text a greater integer occurs, then automatically a transfer to "real" representation is executed which, as a rule, implies that the exactness gets lost.

Further operations on such an interim result can bring it back within integer capacity, but the transfer from real to integer representation is nevertheless refused by the X1, even when the user implicitly wants it, e.g. by assignment to an integer variable, if these arithmetical operations are such that the user is entitled to an exact result: in that case the X1 stops before executing the assignment (stop numbers 104 and 108; see the Appendix). If, however, one (or more) of these operations is one that always delivers its result in real representation (e.g. the normal division) then the assignment to the integer variable is entirely at the responsibility of the user himself, so the X1 will not object against its execution.

The value of a real variable is memorized by the ALP complex in one X1 memory word of 27 bits. (The ALP complex is the only one that has been programmed so far; this description refers to that complex. However, the translator is able to produce object programs that can be executed with a complex memorizing real variables in two-words representation.) Such a value is present in the computer in a so-called floating point representation, i.e. in the form <mantissa> × 2 ↑ <exponent>. Without taking special measures one can count upon a relative precision of ca. five decimal digits (15 bits for the mantissa, 12 for the exponent). However, within certain limits the user can arrange for himself the distribution over mantissa and exponent of the 27 bits of the word. If we call p the number of bits of the exponent, then the greatest absolute value that can be represented is

$$(1 - 2 \uparrow (p - 27)) \times 2 \uparrow (2 \uparrow (p - 1) - 1)$$

and the smallest absolute value is $0.5 \times 2 \uparrow (-2 \uparrow (p - 1))$, and thus zero cannot be represented exactly. The mantissa always is normalized, i.e. abs (<mantissa>) ≥ 0.5 and for the exponent the relations $2 < p \leq 12$ must hold. See also Chapter 5, page 14, sub 7.

In evaluating an arithmetical expression interim results may arise which, in this context, we call "anonymous". The same holds for the value of a function designator. For memorizing anonymous real numbers, always two memory words are available, one for the mantissa, one for the exponent, with the relations $0.5 \leq$ abs (<mantissa>) $\leq 1 - 2 \uparrow (-26)$ and $0 \leq$ abs <exponent> $\leq 2 \uparrow 26 - 1$. This amounts to the following:

An arithmetical expression of "real" type is evaluated with a relative precision of 8 decimal digits (26 bits), but on assignment of the computed value to a real variable up to three decimal digits may get lost and also the "scope" of the numbers is limited. In case a number outgrows the capacity - be it in the computation of anonymous quantities or on assignment to a declared variable - then always the "floating zero" or "infinity" of corresponding precision will come in the place of the unusable result, with the correct sign.

Real constants occurring in the text of the program are memorized, as far as the mantissa is concerned, in "anonymous" precision; for the exponent one has: $-2 \uparrow 11 \leq \langle\text{exponent}\rangle \leq 2 \uparrow 11 - 1$.

For value parameters only, the specifications real and integer come to their full right. For formal parameters that are called by name, these specifications only serve for identifying them as being arithmetical.

Memory Space

The translator occupies about 4200 memory words. The program "Input of Object Program" occupies about 600 memory words. The ALP complex, up to now, consists of about 4000 memory words, but it is memorized only as far as is necessary for the execution of the program.

It is possible for the user to ascertain the amount of memory space occupied by his program and the complex. After storing both into the memory, put the keys of "Begin Address" thus: 00000 00000 11100, and press the button of Autostart 7. Then the typewriter of the X1 will print the desired information. All addresses in the memory that have a higher number are available as "working space", because in executing the program the input program will be overwritten.

How much memory space the program will occupy cannot be estimated with much precision on a simple inspection of the program text. It will be rather safe to count with one memory word for each delimiter. If there are many and/or long strings in the text, more space will be needed, but if there are superfluous brackets and semi-colons space occupation will be less.

Chapter 4

Conventions for Punching

Punching is done on a Flexowriter, Model SFD, especially adapted by the manufacturer to the requirements of the Mathematical Centre. It is an electrical typewriter producing a seven holes punched tape together with a printed copy of the text.

The key TAPE FEED produces blank tape, i.e. tape with sprocket holes only. Every tape must begin and end with at least 25 cm of blank. One can insert blank anywhere in a tape; the X1 will always skip it. However, as soon as the X1 meets blank tape it "forgets" the existing case definition (upper or lower case) and, therefore, after blank a case-dependent symbol must not occur in the tape unless the case has been defined explicitly. Once the X1 has read a case defining symbol this one remains valid until a new case definition is read or until blank is met. All relevant symbols are case-dependent; space, carriage return, tab and the two case defining symbols themselves are not.

One should not confuse the punched symbols space, carriage return and tab (controlling the lay-out of the text printed by the Flexowriter) with the standard operations SPACE, NLCR and TAB occurring explicitly in the program text and controlling the lay-out of the results sheet produced by the typewriter of the X1.

The punchings _ and | do not cause a displacement of the carriage of the Flexowriter. They can be repeated at liberty. They are used for punching underlined symbols, such as begin, go_to etc., and for the

ALGOL symbols:	that are punched:
$\div$	$\underline{:}$
$\uparrow$	$\overset{\wedge}{|}$
$\leq$	$\underline{<}$
$\geq$	$\underline{>}$
$\neq$	ǂ
$\equiv$	$\underline{=}$
$\supset$	$\underline{\rceil}$
$\perp$	space in the text
‘	$\nless$
’	$\ngtr$

A space (also an underlined space) inside an underlined symbol is not allowed. There is one exception, viz. the underlined space in go_to.

Between the two punchings of := a space is not allowed either.

After the end of every program a stop code must be punched.

It is advised to start the text by punching one carriage return and to type it out in a lay-out that will facilitate understanding the "construction" of the program; symbols begin and end corresponding to each other either in the same line or in the same column, labels in column with the begin and end marking off the block to which they are local, etc.

"Erase", i.e. a punching of seven holes, is used for deleting a wrong punching. The X1 will skip it and so will the Flexowriter when used to reproduce the tape.

Numbers in an input tape that are to be read by the MCP "read" are punched in the same way as numbers occurring in an ALGOL program. In the input tape, a number must be followed by a number separator, i.e.:

1. the sign (+ or -) of the next number
2. a tab or at least two spaces
3. carriage return
4. a comma (,)
5. comment to be skipped by "read", i.e. a sequence of ALGOL symbols between apostrophes (')
6. an interrogation mark (?).

"read" considers "inf", "+ inf" and "- inf" as numbers. These make the absolute value of the function designator "read" equal to the greatest real number that can be handled by the complex.

Chapter 5

Directions for Use

When the X1 is in the so-called neutral state, then the keys 0 through 9, G, F and . of the console can be used as autostarts, which means that manipulating one of these keys, or more of them in a fixed order, will start some well defined process inside the computer. The neutral state is established by manipulating key H after the X1 has come to a stop.

For our purpose we use multiple autostarts mainly, viz.:

G 0	Read ALGOL program tape for the first time (= "pre-scan")
G 1	Read same for the second time, translate and produce object program tape
G 2	Read and store object program
G 3	Read complex and store as far as necessary
G 4	Execute program
G 7	Print stop number (and - during pre-scan or tranlation - print line number too).

We shall describe the successive stages of translation and execution of a program in more detail now: Autostart:

1 Put the begin of the translator tape into the tape reader and manipulate in succession the keys H 2 1

2 Put the Flexowriter tape of the ALGOL program into the tape reader and give G 0

3 When the X1 stops, then make certain that stop nr. 1000 has been reached, meaning that the pre-scan has been made correctly. Checking this is easiest by manipulating (G 7)
which autostart causes the stop number and line number to be printed by the typewriter. Any stop number other than 1000 denotes a failure (see the Appendix). Whenever such a failure has been detected the user may try to continue pre-scanning by manipulating the key "Start Next Address", in which case the next stop number may give additional information about the kind of the mistake. (The brackets around G 7 mean that there is no obligation for using this autostart: if the user is more familiar with the properties of the X1, then he can easily read the stop numbers from the register OR (= Order Register).)

Autostart:

4 If the pre-scan was successful, then engage the tape punch, put the begin of the ALGOL program tape into the tape reader again and give G 1

(Be sure that a. there is enough tape in the tape punch
b. the first and the last key of the console word are set downward.)

After translating ask for stop number 1001 with (G 7)

The X1 has punched an object program tape now. If the user wants so, he may at once translate some other ALGOL program without needing to restore the translator.

5 The execution of every program must be preceded by storing the program "Input of Object Program". Put the tape thereof into the tape reader and give H 2 1

6 Put into the tape reader that end of the object program tape that was punched l a s t and give G 2

7 Make sure that stop number 14 has been reached, by (G 7)

Now, the user is given the opportunity for communicating to the computer the number of bits, p, that he desires to be available for memorizing the exponent of a real variable ($2 < p \leq 12$, see also Chapter 3, page 9, 3rd paragraph). To do so, give (+ p G)
If this is omitted, then the X1 makes $p = 12$.

8 Put the tape containing the complex into the tape reader and give G 3

9 Ask for stop number 12, by giving (G 7)

(If the stop has another number and the tape has not yet been read completely then most probably X1 made a reading error. In such a case put under the tape reader the piece of blank that passed it last and set the keys of "Begin Address" in the positions: 11000 01001 00000; manipulate the key "Start Begin Address". If this does not give a satisfactory result then go back one more piece of blank and try again.)

Autostart:

10 Make all preparations necessary for the execution of the program:

a. Are the keys of the console word to be set?

b. Is there an input tape to be read by the program? If so, put it under the tape reader.

c. Should the typewriter be engaged? If so, should tabs be set? Or distance of lines? Is there enough paper in it?

d. Should the tape punch be engaged? If so, is there enough tape in it?

If everything is in good order, give G 4

11 If the program stops, then ask for stop number 110 (= End of Program) or 115 (= Standard Procedure "stop" in the program), by giving (G 7)

If one only wants to use the translator for checking whether the ALGOL text is correct grammatically, then take only the steps 1 through 4 of the above, but the key that is most to the left in the console word must then be set upward before giving G 1. In that case the punching of an object program tape is suppressed. However, by this procedure the checking is not as thorough as it can be. Only when a program is executed the last checks are made.

One should keep in mind that it will always remain possible to compose a sequence of ALGOL symbols that - if presented as being an ALGOL program - leads the translator astray but will cause the X1 to stop in an unexpected way.

Appendix: The Stop Numbers

The stop numbers (typed out by autostart G 7 or read directly from the Order Register) contain hints about the probable cause of stopping; below we give a complete list of them.

In the explanations given, a symbol followed by the words "not appropriate" means: the symbol involved is not allowed after the symbol(s) preceding it. Where a stop number is said to be non-interpretable, the most probable explanation is that the X1 has made a mistake.

Stop number	A During pre-scan and translation:
1000	end of pre-scan
1001	end of translation
1002	wrong parity or "non-existing" punching
1003	case definition missing
1004	| is followed by illegitimate symbol
1005	underlining missing
1006	delimiter composed of insufficient number of symbols
1007	a symbol ' or " occurs outside strings
1008	comment in an illegitimate place
1009	) is followed by a cipher or by some letters and ciphers
1010	)<letter string> is not followed by :
1011	)<letter string> : is not followed by (
1012	<identifier> or true or false is followed by . or $_{10}$
1013	true or false is followed by a non-permissible symbol
1014	a number is followed by a letter
1015	$_{10}$ is not followed by + or - or <integer>
1016	stop code or ? in an illegitimate place
1017	a formal parameter list contains more than 31 identifiers
1018	⊁ precedes ⊀
1019	failure of X1
1020	? or stop code in an illegitimate place
1021	switch identifier in switch declaration is not followed by :=
1022	illegitimate ALGOL symbol or non-ALGOL symbol
1023	<procedure identifier> illegitimately not followed by ; or (
1024	<procedure identifier> (.... is not followed by)
1025	<procedure identifier> (...) is not followed by ;
1026	identifier or number missing
1027	the number of (round or square) brackets in a statement is not correct
1028	own in an illegitimate place
1029	own illegitimately not followed by real or integer or Boolean
1030	declaration not followed by ;
1031	declaration in an illegitimate place
1032	specificator or declarator in an illegitimate place
1033	in a specification some delimiter other than , or ; occurs
1034	array declaration is not correct
1035	array declaration is not followed by ;
1036	non-interpretable stop

Stop number	
1037	begin not appropriate
1038	+ or - not appropriate
1039	× or / or ÷ or ↑ not appropriate
1040	number of bound pairs or actual parameters is not correct
1041	failure of X1
1042	go to is not followed by a designational expression
1043	Boolean expression missing
1044	arithmetical expression missing
1045	in the procedure body a non-specified parameter is called
1046	identifier specified to be a string occurs in a way different from being an actual parameter
1047	in procedure statement the actual parameter list is missing
1048	first occurrence of identifier in an illegitimate place
1049	in the procedure body of a function procedure no assignment of a value to the procedure identifier occurs
1050	on the two sides of else expressions occur of incompatible type or kind
1051	end not appropriate
1052	failure of operator: autostart is used incorrectly
1053	number of "nested" blocks greater than 31
1054	; not appropriate
1055	: not appropriate
1056	go to not appropriate
1057	value or string or switch or integer or real or Boolean or label not appropriate
1058	own or integer or real or Boolean not appropriate
1059	own in procedure heading
1060	switch not appropriate
1061	array or procedure not appropriate
1062	(not appropriate
1063	, not appropriate
1064	) not appropriate
1065	:= is preceded by non-permissible left part variable
1066	:= not appropriate
1067	< or ≤ or = or ≥ or > or ≠ not appropriate
1068	¬ not appropriate
1069	∨ or ∧ or ⊃ or ≡ not appropriate
1070	if not appropriate
1071	then not appropriate
1072	else not appropriate
1073	≮ not appropriate
1074	for not appropriate
1075	:= not appropriate in for-statement
1076	do not appropriate
1077	step not appropriate
1078	the actual parameter of a standard function is not an arithmetical expression
1079	until not appropriate
1080	while not appropriate

Stop number	
1081	, not appropriate in for-statement
1082	:= not appropriate in switch declaration
1083	, not appropriate in bound pair list
1084	, not appropriate as array segment separator
1085	; not appropriate in array declaration
1086	[not appropriate
1087	, not appropriate as subscript separator
1088	] not appropriate
1089	function procedure missing
1090	element of switch list is not a designational expression

<u>B</u> During activity of program "Input of Object Program":

0	non-interpretable stop
1	operator used an autostart that is not permissible now
2	non-interpretable stop
3	operator used non-existing autostart
4	MCP is programmed incorrectly: number > 127 under DWI
5	" " " " : number > 31 under DWP
6	" " " " : number > 63 under DWS
7	failure of tape reader
8	MCP is programmed incorrectly: combination DWD non-permissible here
9	pentade > 9 instead of a decimal digit. Failure of tape reader?
10	non-permissible directive in tape
11	MCP is programmed incorrectly: a constant outgrows capacity
12	end of tape containing the complex
13	wrong parity in object tape. Failure of tape punch, or of tape reader?
14	end of object program tape

<u>C</u> During execution of object program:

100	operator used an autostart that is not permissible now
101	non-interpretable stop
102	operator used non-existing autostart
103	the problem outgrows the capacity of the memory
104	interim capacity overflow in integer arithmetic
105	capacity overflow in integer arithmetic
106	failure of typewriter. Make a note that last symbol is printed incorrectly and manipulate key "Start Next Address"
107	non-permissible actual parameter
108	in typing or punching an interim capacity overflow in the integer arithmetic is detected
109	the program is writing outside the capacity of an array
110	end of program
111	overflow in array administration: bound(s) too great

Stop number	
112	the number of dimensions of an array occurring as an actual parameter is incorrect
113	the number of parameters of a procedure occurring as an actual parameter is incorrect
114	integer parameter of SPACE missing
115	procedure "stop" in program
116	the operand of an integer division is of real type
117	integer parameter of XEEN missing
118	integer parameter denoting number of digits before or after the decimal point in FIXT missing
119	capacity overflow in "entier"
120	procedure "read" finds a stop code
121	procedure "read" does not find a case definition
122	procedure "read" finds an unknown upper case symbol
123	procedure "read" finds an unknown lower case symbol
124	as 111, now for own arrays
125	integer parameter denoting number of digits of the mantissa in FLOT missing
126	integer parameter of EVEN missing
127	as 118, now for ABSFIXT
128	integer parameter of SUM missing
129	actual parameter of SETRANDOM does not fulfill the conditions $0 \leq a < 1 - 2 \uparrow (-27)$
130	punching routine with non-permissible parameter
131	integer parameter for PUSPACE missing
132	integer first or second parameter of FLOP missing
133	" " " " " " FIXP "
134	" " " " " " ABSFIXP "
135	actual parameter of PUTEXT is not a string
136	actual parameter of PU7BIT is not an integer fulfilling the conditions $0 \leq a < 128$

Concluding Remarks

Now, after a year of experience with the system, it is possible to signal some deficiencies we shall try to avoid in later projects:

1. Though it has been written as a complete compiler system, it was used mainly as a text checker until now. For this latter purpose a one-pass program would be preferable.

2. Obvious possibilities for detecting more than one mistake at a time have been neglected. Mr. Felsch, of the Chr. Albrecht Univ., Kiel, effected a substantial improvement in this respect by saving and restoring the links and in the subroutine "type stop number and line number" of the translator (see [4], page 131).

3. Mistakes against 2.4.3 of [1] (i.e. using the same identifier twice with non-disjunct scopes) are checked insufficiently.

We wish to express our gratitude to Mr. Ch.Harmse for his help in preparing this report.