1130 ALGOL USERS MANUAL

This document is written for the general user of the compiler. The reference language is described together with the conventions for writing programs and the use of the input-output procedures. Information is provided as to the use of the compiler under Monitor. A list of error messages generated by the various phases is provided.

A second document (The internal working of the 1130 ALGOL compiler) is provided for those who wish to write code procedures and for those who wish to amend the compiler. It contains a detailed description of the workings of the compiler, the method of translation and all the conventions of coding and transmission.

The 1130 ALGOL compiler requires the following minimum machine configuration:

- 8K core (the compiler can also work with 16K)
- disk
- input device - card reader, paper tape reader or console keyboard
- output device - printer, paper tape reader or console printer

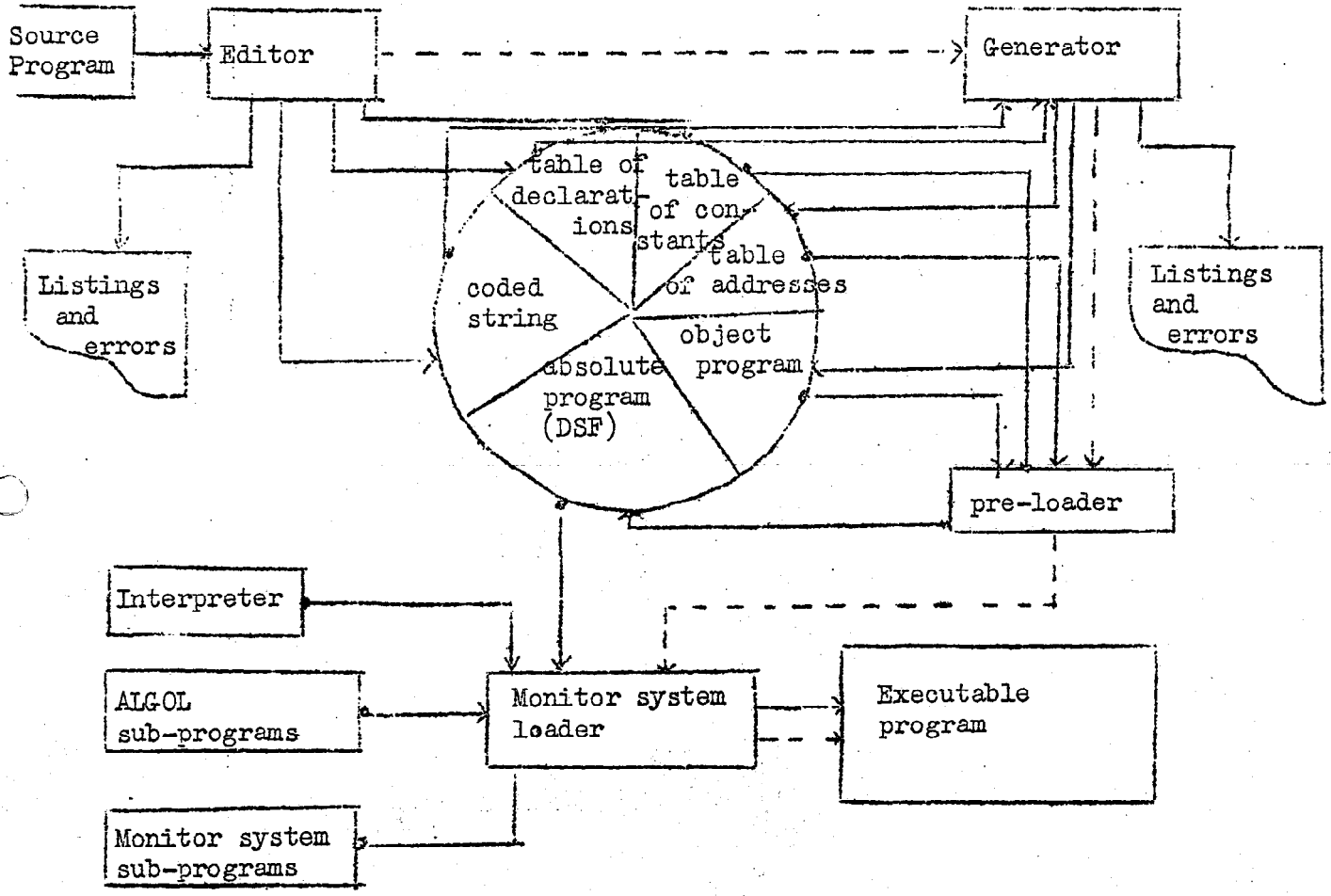
INDEX

1. The structure of the compiler

The compiler proper consists of two parts:

- the editor - which reads the source program from cards, paper tape or keyboard, optionally produces a program listing and stores the coded string onto disk together with the initial translation of the program, the table of constants and the table of declarations.
- the generator - which uses the coded string and the table of declarations to construct the object program which is also stored on disk. Optionally a listing of the object program can be obtained. To make it acceptable to the system the object program is next processed by
- the pre-loader - which combines it with the table of addresses and the table of constants to an absolute program in Disk System Format (DSF).

This object program is not directly executable. It is interpreted by a third part of the compiler, the interpreter which is loaded onto core at the same time as the object program.



2. The reference language definition

The principal restrictions with reference to the ALGOL 60 Report of the language recognised by the compiler are the following:

- All formal parameters of procedures must be specified.
- Labels cannot be unsigned integers.
- The declarator own does not exist.
- A goto instruction that points to an undefined switch is treated as an error.
- A formal array parameter cannot appear in the value part.
- Only the first five characters of an identifier are significant.
- Only capital letters are available in identifiers.

More precisely, the restrictions are defined below in terms of the ALGOL 60 Revised Report.

2.1 Delete in the definition of < letter > ;
"a/b/.... /y/z";

2.3 Delete in the definition of < declarator > :
"own"

2.4.3 Add in the third sentence ("They may be chosen freely.."):
", but there is no effective distinction between two different identifiers whose first five symbols are identical."

3.5.1 Delete in the definition of < label > : " < unsigned integer > ".

3.5.5 Delete this paragraph.

4.3.5. Replace: "equivalent to a dummy statement" by "undefined".

4.7.5.3 Replace the second sentence by "It cannot be called by value".

4.7.5.5 Delete this paragraph.

5. Delete the first two sentences of the fourth paragraph.

5.1.1 Delete the definition of < local or own type > :
replace the definition of < type declaration > by:
" < type declaration > " :: = < type > < type list > ."

5.1.3 Delete the last sentence.

5.2.1 Replace in the definition of < array declaration >
" < local or own type > " by " < type > ".

5.2.2 Delete the second example.

5.2.5 Delete this paragraph.

5.4.5 Replace the third sentence by:
"All the formal parameters must be specified."

3. Writing Conventions

All legal EBCDIC characters are recognised if the program is read from cards. From paper tape or console keyboard only the characters of the sub set are recognised. In all cases, the characters ' (apostrophe) and @ (commercial at) are equivalent. Only the first four letters of the basic symbols between apostrophe are signficant. Thus 'BOOL' and 'BOOLEAN' are equivalent. Spaces are only significant within strings. Thus 'GO TO' and 'GOTO', (/ and (/ are equivalent.

The various notations recognised for each basic symbol and shown in the following table can be used together without any restrictions.

The physical representation of basic symbols

ALGOL Notation	Standard Notation	Other Notations
+	+	
-	-	
X	*	
/	/	
÷	'/'	

ALGOL Notation	Standard Notation	Other Notations
↑	**	'PØWER'
<	'LESS'	< 'INF'
≤	'NØTGREATER'	<= 'ING'
=	'EQUAL'	= 'EGAL'
≥	'NØTLESS'	>= 'SUC'
>	'GREATER'	> 'SUP'
≠	'NØTEQUAL'	¬= 'DIF'
≡	'EQUIV'	
⊃	'IMPL'	
V	'OR'	1 'ØU'
Λ	'AND'	& 'ET'
¬	'NØT'	¬ 'NØN'
goto	'GØ TØ'	'ALLER A'
if	'IF'	'SI'
then	'THEN'	'ALØRS'
else	'ELSE'	'SINØN'
for	'FØR'	'PØUR'
do	'DØ'	'FAIRE'
boolean	'BØØLEAN'	'BØØLEEN' 'B'
integer	'INTEGER'	'ENTIER' 'I' 'E'
real	'REAL'	'REEL' 'R'
array	'ARRAY'	'TABLEAU'
switch	'SWITCH'	'AIGUILLAGE'
procedure	'PRØCEDURE'	
string	'STRING'	'CHAINE'
label	'LABEL'	'ETIQUETTE'
value	'VALUE'	'VALEUR'
10	'	
:	..	:
:=	.=	:=
;	.,	;
step	'STEP'	'PAS'
until	'UNTIL'	'JUSQUA' 'A'
while	'WHILE'	'TANT QUE'
⌈	(/	
⌋	/)	
6	'('	
9	')'	

ALGOL Notation	Standard Notation	Other Notations
begin	'BEGIN'	'DEBUT'
end	'END'	'FIN'
true	'TRUE'	'VRAI' 'T' 'V'
false	'FALSE'	'FAUX' 'F'

4. Input/output Procedures

All data transmissions between an ALGOL program and an peripheral device must be done by procedure call. The following description concerns the standard procedures provided with the compiler.

A. Standard procedure identifiers

All these procedures, except for SYSACT, have an english name and a french name. Both names of the same procedure can be used freely within the same program and they will always refer to the same procedure. All the standard procedures are considered to be declared in a fictional block containing the program.

They are in four catagories:

	Procedure identifier	French equivalent
Input procedures	INSYMBOL	ENTSYMBOLE
	INREAL	ENTREEL
	ININTEGER	ENTENTIER
	INBOOLEAN	ENTBOOLEEN
	INARRAY	ENTRTABLEAU
	INTARRAY	ENTETABLEAU
	INBARRAY	ENTBTABLEAU
Output procedures	OUTSYMBOL	SORSYMBOLE
	OUTREAL	SORREEL
	OUTINTEGER	SORENTIER
	OUTBOOLEAN	SORBOOLEEN
	OUTSTRING	SORCHAINE
	OUTARRAY	SORRTABLEAU
	OUTTARRAY	SORETABLEAU
	OUTBARRAY	SORBTABLEAU
Control procedures	SYSACT	-

Procedure identifier	French Equivalent
Procedures for intermediate storage	
PUT	METTRE
GET	PRENDRE

B. Input/Output devices

The first parameter of all the standard procedings for PUT and GET, is an integer whose value denotes the type of input/output device concerned. Transmission between the program and peripheral is always one way.

All peripherals used by the program must be defined before compilation by a control card *E/S or *IOCS (see Section 5). The numbers associated with the peripherals are as follows:

Ø Card Reader.

1 Printer.

2 Paper Tape Reader.

3 Paper Tape Punch.

4 Console Keyboard.

5 Console Printer.

6 Card Punch.

If the value of the parameter is negative or greater than 6 or corresponds to a device not defined by a *E/S or *IOCS card this will cause an error at program execution.

C. The simple input/output procedures

Each device has associated with it

a record pointer, S,

a character pointer, R,

a record length, P, and in the case of the printer

a page length, Q.

When each character is transmitted, R: = R + 1
When R = P then S: = S+ 1 and R: = 1 (i.e. the character pointer is reset). The increase in S implies the physical transmission of a record.

In the case of the printer, if the record pointer, S, which is equivalent to a line of output, equals the value Q, a new page is started and S is reset to 1.

Also each device has a number, K, associated with it. This indicates the number of spaces which can be used as a delimiter between numbers.

The initial values of all these internal parameters are

shown below:

Device		Character Pointer, R	Record Pointer, S	Record Length, P	Page Length, Q	Space Delimiter K
Ø	Card Reader	80	1	80	*	2
1	Printer	1	1	120	∞	2
2	Paper Tape Reader	80	1	80	*	2
3	Paper Tape Punch	1	1	80	*	2
4	Keyboard I/P	120	1	120	*	2
5	Keyboard O/P	1	1	120	*	2
6	Card Punch	1	1	80	*	2

a) Procedures INSYMBOL and OUTSYMBOL

Their specifications are:

```
procedure INSYMBOL (N, STR, D); value N; integer N, D;
string STR;

procedure OUTSYMBOL (N, STR, S); value N, S; integer N, S;
string STR;
```

The first parameter defines the device concerned; the second parameter is a string which is used as a dictionary: the characters are numbered from left to right starting from 1. By means of this numbering, the procedure INSYMBOL assigns to its third parameter the number corresponding to the appearance in the string of the character read in from the device in question. If this character does not appear in the string, the value returned is zero. Conversely, the procedure OUTSYMBOL sends the appropriate device the character, whose position in the string corresponds to the value of the third parameter. If this third parameter is Ø or greater than the length of the string, the procedure transmits a blank.

Finally, both procedures increase the character pointer by one.

Example:

```
for 1: = 1 step 1 until 15 do
begin
INSYMBOL (Ø, 'ABCDEFGHIJKL', V);
OUTSYMBOL (1, '123456789Ø+', V);
end;
```

If the following character string appeared on a card during imput:

... ...AXBIDA+4EFMJ5FK..., this example would output:
... 1.2941ww56ø6+...

b) Procedures INREAL and OUTREAL

Their specifications are:

procedure INREAL (N, D); value N; real D;

procedure OUTREAL (N, S); value N, S; integer N; real S;

The procedure INREAL scans the input character string from the relevant device until it finds a number written according to the ALGOL definition. During this scan the first syntactically incorrect character met is used as the delimiter. The end of a record or a succession of K consecutive spaces also serve as delimiters. If the collection of characters found before one of these delimiters does not form a syntactically correct number, the scan starts again.

The procedure OUTREAL outputs the value of its second parameter to the relevant device in the following standard format: ± X.XXXXXXXX ± YY where the X are the significant digits and YY the exponent. After the number the procedure outputs K spaces (unless there are less than K characters before the end of the record).

Example:

```
for i: = 1 step 1 until 5 do
begin
INREAL (ø, V);
OUTREAL (1,V)
end;
```

If the following string appeared on a card during input:
...1,-wwø34.5w'5ABC + -' 7.Aøw - '1X..., the example would transmit (assuming K = 2):

... + 1.øøøøøøøø ' + øøww+ 3.45øøøøøø + ø6ww- 1.øøøøøøøø' + ø7ww
+ ø.øøøøøøøø'+ øøww- 1.øøøøøøøø ' +ø 1

c) Procedures ININTEGER and OUTINTEGER

Their specifications are:

procedure ININTEGER (N,D); value N; integer N, D;

procedure OUTINTEGER (N, S); value N, S; integer N, S;

The action of these procedures is identical to INREAL and OUTREAL except that the number read in by ININTEGER is converted to type integer and the number output by OUTINTEGER is aligned to the right in a 6 character field preceded by a sign.

d) Procedures INBOOLEAN and OUTBOOLEAN

Their specifications are:

procedure INBOOLEAN (N, D); value N; integer N; boolean D;

procedure OUTBOOLEAN (N, S); value N, S; integer N; boolean S;

These procedures work in the same way as the preceding ones. INBOOLEAN searches for the characters 'TRUE' and 'FALSE' OUTBOOLEAN outputs one or other of these symbols. The treatment of blanks is the same as for the preceding procedures.

e) Procedure OUTSTRING

Its specification is:

procedure OUTSTRING (N, STR); value N; integer N; string STR;

This procedure transmits one by one the characters in the string which make up the second parameter; the text thus formed can span over several records. It is not terminated by K blank separators.

Example OUTSTRING (1, 'ALGOL␣'); OUTSTRING (1, 'REPORT');
This would transmit to the output line the following characters:

... ALGOL␣REPORT ...

D. Procedures for array transmission

Their specifications are as follows:

procedure INARRAY (N, D); value N; integer N; array D;

procedure OUTARRAY (N, S); value N; integer N; array S;

procedure INTARRAY (N, D); value N; integer N; integer array D;

procedure OUTTARRAY (N, S); value N; integer N; integer array S;

procedure INBARRAY (N, D); value N; integer N; boolean array D;

procedure OUTBARRAY (N, S); value N; integer N; boolean array S;

These procedures transfer the complete contents of the array which is their second parameter, by repetative calls of their corresponding simple procedures.

Example If an array T is declared: "T[I1 : S1, I2 : S2, ...]", the instruction: "INARRAY (Ø, T);" is equivalent to the block:

```
begin integer K1, K2, ...;
    for K1: = I1 step 1 until S1 do
    for K2: = I2 step 1 until S2 do
    INREAL (Ø, T[ K1, K2, ... ])
end;
```

E. Control Procedure

The procedure SYSACT allows access or modification to "system parameters" associated with each input/output device. The system parameters are as follows:

R character pointer.

S record pointer.

P record length.

Q number of lines per page (printer only)

K number of blank separators.

The specification of the procedure SYSACT is as follows:

procedure SYSACT (N, F, Quantity); value N, F; integer N, F, Quantity;

The first parameter specifies the device concerned.

The second parameter specifies the nature of the required operation according to the table below.

Value of F	Action by SYSACT
1	Quantity : = R.
2	R : = Quantity. R must be less or equal to P. If Quantity is less than present R then the pointer goes to character, R of record S +
3	Quantity : = S.
4	Same effect as for F = 14.
5	Quantity : = P.
6	P : = Quantity; P must ≤ to the initial value.
7	Quantity : = Q } No effect except for the
8	Q : = Quantity } printer.
9	Quantity : = K.
10	K : = Quantity
11	No effect. Only applicable on 360.
12	No effect. Only applicable on 360.
13	Error.
14	Jump "Quantity" records.
15	Jump to the head of the next page (no effect except on the printer), then the same effect as for F= 14.

F. Procedures for intermediate storage

The two procedures PUT and GET allow the use of the disk as work space during the execution of a program.

Their specifications are as follows:

procedure PUT (N, LIST); value N; integer N; procedure LIST;

```
procedure GET (N, LIST); value N; integer N; procedure LIST;
```

The first parameter is a sector number on disk (the number relative to the start of the work area).

The second is a list procedure which is used to communicate with the elements to be transmitted. In the body of this procedure, each element to be transmitted appears as a call of the procedure given as the parameter of the list procedure.

Example: To transmit onto sector 10, the values of A, B and C, one can write:

"PUT (1Ø, LIST);" the procedure list having the declaration:

```
procedure LIST (SEND); procedure SEND;

begin
SEND (A); SEND (B); SEND (C)
end;
```

To insert into array T, previously written values starting from sector N one can write : "GET (N, ARRAY);" the procedure ARRAY having the declaration:

```
procedure ARRAY (ELEMENT); procedure ELEMENT;

begin
integer I;
for I : = 1  step 1  until M  do ELEMENT (T[ I ] )
end;
```

or even, more simply

```
procedure ARRAY (ELEMENT); procedure ELEMENT;
          ELEMENT (T);
```

Each call of GET or PUT starts transmission automatically at the begining of a sector. Transmission can continue over several consecutive segments. Thus it is important to keep count of the actual number of machine words transmitted by each call. An integer or boolean each occupy one word, a real quantity two words and a string two characters per word.

The following list procedure will transmit 76 words:

```
begin
integer array A [ 1 : 2Ø ] ; array B[ 1 : 1Ø ] ;
boolean P, Q;
procedure LIST (FETCH); procedure FETCH;

    begin
    integer I;
    for I : = 1 step 1 until 10  do
```

```
        begin
        FETCH (B[ 1, I ]); FETCH (B[ S, I ])
        end;
    FETCH (P); FETCH (A); FETCH (Q);
    FETCH ('␣SMALL␣STRING␣OF␣CHARACTERS.')
end;
```

5. The practical use of the compiler

The ALGOL compiler is inserted in the general organisation of the 1130 Monitor System. Once put onto disk, it is used in the same way as the FORTRAN compiler or ASSEMBLER.

The compiler is called by the control card: // XEQ ECIAL

Othercontrol cards may appear between this first card and the program to be compiled. They are recognised by an asterisk in column 1 and may be any of the following:

*LIST SOURCE PROGRAM or

*LISTER PROGRAMME SOURCE

When reading in the program from the paper tape, this card causes a program listing to be output to the printer during the editor phase. Each line is preceded by the semi colon count at the end of the previous line.

*LIST IDENTIFIERS or

*LISTER IDENTIFICATEURS

This card requests a listing of the table of identifiers on the system output device (1132 printer or the console printer) at the end of each block.

*LIST OBJECT PROGRAM or

*LISTER PROGRAMME OBJECT

This card requests a program listing on the system output device during generation.

*LIST ALL or

*LISTER TOUT

This card replaces the three preceding ones.

*NAME XXXXX or

*NOM XXXXX

This card defines the name of the generated program. (Any 5 characters in columns 6 to 10 or 7 to 11).

** Header Information:

The information contained between columns 3 to 80 of this card is printed at the head of every page of listing

of the source program.

* IOCS (PRINTER, CONSOLE, CARDS, PAPERTAPE) or

* E/S (IMPRIMANTE, PUPITRE, CARDS, RUBAN)

This card defines the input/output devices used by the object program. Only enter those devices actually used as this avoids the unnecessary loading of sub programs and the reservation of unused work areas.

The program to be compiled must be terminated by the card: / /*FAL

When this card is read, the editing phase is terminated. If no errors have been detected, the compiler then passes to the generation phase, which finishes all being well with the message "BONNE COMPILATION". This message means that the program has been compiled but it does not mean that there have been no errors. The pre-loader comes last and then the compiler returns control to the system monitor, at which point the object program is in the work area on disk. It is then possible to directly execute the program by inserting an XEQ card or to save it by a DUP card.

As the utility sub program used by the object program is always DISK0, the XEQ card must have a zero in column 19.

Examples:

```
/ /*JOB
// XEQ                                    ECIAL
*IOCS (PRINTER, CARDS)
*LIST SOURCE PROGRAM
*LIST IDENTIFIERS
*NAME TEST
          'BEGIN'
                ALGOL PROGRAM
          'END'
/ /*FAL
/ /*DUP
* STORE WS UA TEST
/ /*XEQ TEST L Ø1 Ø
* LOCAL TEST, PRNT 1, CARD Ø, HOLE B
          .
          .
          .  program data
          .
          .
/ /*JOB T
// XEQ
          'BEGIN'
            .
            .
          'END'
```

```
/ /WFAL
/ /WXEQ          ø
      .
      .
      . data
      .
      ,
/ /WJOB
```

6. Error Messages

Errors can be deleted at three different times:

a) during the editing phase (reading the source program),

b) during generation (producing the object program),

c) during interpretation (executing the object program).

The error messages produced in the three cases are very different in nature and significance. Their common characteristic is that they output the current semi colon count which allows the user to pinpoint the place where the error occured.

A. Error messages from the editor

These are printed onto the system output device in the following format:

"ERREUR xx *PV* yyyy "

where xx is the error number

yyyy is the semi colon count

PV is "point-virgule" (semi colon).

The table below is a list of the existing error numbers, the reason for failure and the resultant action of the editor. The errors marked with an asterisk are not considered serious enough to prevent the generation of the object program. Errors marked with a double asterisk are catastrophic and will cause an immediate stop in translation and will return control to the Monitor System. The other errors do not stop the editing phase but will stop generation of the object deck.

Number	Type of error	Action taken by editor
01	Illegal character in a compound basic symbol	taken as a terminating apostrophe.
02	Illegal character in a comment between procedure parameters.	taken as the end of the comment.
03	Unrecognised character	ignored
05	Unrecognised compound basic symbol	ignored
07	No apostrophe after '(	following character taken as '

Number	Type of error	Action taken by editor
08	Apostrophe out of context	ignored
09	No apostrophe after /	following character taken as '
10 **	/ / card out of context	fatal
11	Illegal character after period	ignored
12	More than one decimal point in a number	ignored
16 *	Integer constant > 32767 or < - 32768	taken as real
17 **	Overflow in table of constants	fatal
18 *	Real constant too large (> 10^{38})	Maximum taken
19 *	Real constant too small (> 10^{-38})	Minimum taken
20 **	Too many semi colons in a program	fatal
40 **	Stack overflow (declarations and bracketing of instructions and expressions)	fatal
41 **	Too many identifiers	fatal
42	Multiple declaration	ignored
43 **	Too many declarations in one block (> 123 words where an integer, a boolean or an array require one word and a real two)	fatal
44 **	Too many labels, switches or procedures (total > 1024)	fatal
47	Extra end	ignored
48 **	Blocks rested too deep (> 16)	fatal
49 **	Too many parameters in a procedure (> 41)	fatal
50	end,),], or ; missing	symbol assumed
51	No identifier after switch	Jump to next semi colon
52	No : = in a switch declaration	as for 51
53	integer, real or boolean followed by a compound basic symbol other than array or procedure	as for 51
54	Identifier missing from a declaration	as for 51
55	Separator missing from a declaration	as for 51
56	Separator other than; or , in a declaration	taken as ;

Number	Type of error	Action taken by editor
57	begin ommitted	assumed
58	Declarator out of context	ignored
59	(missing	assumed
60	Separator missing after] in an array declaration	-
61	Separator other than ; or , in an array declaration	as for 56
62	[missing	assumed
63	] or) missing	) inserted
64	Separator other than, or [in an array bound pair list	as for 51
65	No) after a procedure identifier	as for 51
66	Identifier missing from a parameter list	as for 51
67	Separator other than , or) in a parameter list	taken as)
69	; missing after) in a procedure declaration	assumed
70	Identifier missing after code	code ignored
71	Specification of an identifier which is not a parameter	ignored
72	Multiple specification of an identifier	ignored
73	No specification for a parameter	parameter assumed integer
99	Illegal control card	ignored

B. Error messages from the generator

These are printed on the system output device in the following format:

"ERREUR AAAA
PV NNNN
ETAT = XX, SYMB = T N (BB) ZZZ"

where

AAAA is the type of error (syntactic, semantic or type),

NNNN is the current semi colon count,

XX is the current state.

T, N and ZZZ indicate the nature of the current syntactical unit. BB only appears if this unit is a declared identifier.

The program logic manual 'La Notice de Fonctionnement' contains the meaning of the various codes appearing in the message.

In general if T = 4 and N = 3 the syntactic unit is a basic symbol (not an identifier or constant) and ZZZ describes this basic symbol according to the following table.

Number	Symbol
1	+
2	−
3	− unary
4	×
5	/
6	÷
7	↑
8	≤
9	<
10	=
11	≥
12	>
13	≠
14	¬
15	∧
16	∨
17	⊃
18	≡
19	goto
20	if
21	then
22	else
23	for
24	do

Number	Symbol
25	,
28	:
29	;
30	; at the end of a procedure
31	: =
33	step
34	until
35	while
37	(
38	)
39	[
40	]
41	'
45	begin
46	end
50	array
51	switch
52	procedure
56	code

A semantic error is an error detected by an empty element in the decision table.

A type error corresponds to a type incompatibility of an identifier or expression with the current situation.

A syntactic error is one detected in any other way.

In addition the generator detects non-declaration of

identifiers by the message "IDENTICATEUR NON DECLARE". ZZZ contains the static number of this identifier which corresponds to that provided by the editor in the table of identifiers. The errors,

ERREUR DISQUE	(Disk fault),
DISQUE DE BORDE	(Disk overflow),
PILES DE BORDEES	(Stack overflow),
SOUS-PASSEMENT PILE	(Stack underflow),
ERROR SYSTEM	(System fault),

speak for themselves and always cause a catastrophic failure.

C. Error messages during execution

The detection of an error during the execution of the program causes a small final phase to be loaded which prints out on the system output device the appropriate error message accompanied by the current semicolon count in the following format

ERREUR DETECTEE PAR xxxx AU POINT-VIRGULE yyyy zz.....z

xxxx is the program name which has found the error.

e.g. INTAL if it is the interpreter

EREAL if it is a standard function etc.

yyyy is the semi colon count

zz ....z is the appropriate error message.

After outputting these messages, the 1130 stops. If the operator sets the keyboard keys to a non zero number and restarts the user can obtain a hexadecimal dump of the execution stack which contains all the variables and arrays being used by the program.