ELLIOTT 900

Volume 2: PROGRAMMING INFORMATION

Part 1: PROGRAMMING LANGUAGES

Section 2: ALGOL

Contents

		Page
	Preface	iv
Chapter 1:	SYSTEM SPECIFICATION	
	1.1 Elliott 903 Algol Representation	1
	1.1.1 Basic Symbols	2
	1.1.1.1 The Characters erase, runout and halt	2
	1.1.2 Punching Instructions	3
	1.1.3 The Use of Single Case	4
	1.1.4 Notes to the Programmer	5
	1.1.5 The Program	5
	1.1.6 The Use of Elliott Algol Program Sheets	7
	1.1.7 Correction of Algol Programs	7
	1.2 Restrictions and Programming Notes	8
	1.2.1 Restrictions	8
	1.2.1.1 The Declaration of Labels	8
	1.2.1.2 Type of Arithmetic Expressions	10
	1.2.1.3 for statements	10
	1.2.1.4 own declarations	10
	1.2.1.5 Specification of Parameters	10
	1.2.1.6 Recursive Procedures	10
	1.2.1.7 Parameters	11
	1.2.1.8 Type Procedures	11
	1.2.1.9 Sequence of Declarations	11
	1.2.1.10 Length of Identifiers	11
	COMMENT 1.2.2.4 (3)	14

Page

1.2.1.11 Reserved Identifiers 11
1.2.1.12 Limits 11
1.2.2 Programming Notes.. 12
1.2.2.1 Range and Precision of Numbers 12
1.2.2.2 Sequence of Operations 13
1.2.2.3 Boolean Expressions 13
1.2.2.4 Correspondence Between Formal and Actual Parameters 14
1.2.2.5 goto if 15

Chapter 2: INPUT AND OUTPUT
2.1 Introduction 16
2.1.1 Print and Read Statements 16
2.1.2 Structure of Read and Print Lists.. 16
2.1.3 Input Data Tape 17
2.1.4 Output of Text 18
2.1.5 Output of Text 19
2.2 Setting Procedures 20
2.2.1 Device Setting Procedures 20
2.2.2 Prefix Setting Procedures 21
2.2.3 Format Setting Procedures 21
2.2.4 Alarm Printing 22
2.2.5 Parameters out of Range 23
2.2.6 Input and Output of Strings 23
2.2.7 Procedures in Read and Print Lists 24

Chapter 3: STANDARD PROCEDURES
3.1 Introduction 25
3.2 Algol Standard Functions 25
3.2.1 real Procedures 25
3.2.2 integer Procedure sign(E) 25
3.2.3 Entier 25
3.3 Checking Functions 26
3.4 Machine Code 27
3.4.1 Declaration in Algol text 27
3.4.2 As written in SIR 28
3.4.2.1 Scalar Parameters 29
3.4.2.2 Result of a Type Procedure 29
3.4.3 Rules for SIR Blocks 31
3.4.4 Array Parameters 32
3.4.5 Label Parameters 34
3.4.6 Switch Parameters 35
3.4.7 String Parameters 36
3.4.8 Use of Interpreter Subroutines 36
3.4.9 Real Parameters 37
3.4.9.1 Unpacking.. 37

			Page
	3.4.9.2	Packing	37
	3.4.9.3	real Parameters called by name	38
	3.4.9.4	Integer to real conversion	38
	3.4.9.5	Real to integer conversion	38
	3.4.10	User of interrupts in SIR code procedures	38
	3.4.11	Loading Machine Coded Procedures	40
	3.4.11.1	As an extension to the library	40
	3.4.11.2	At Run Time	40
3.5	The Library		40
	3.5.1	Structure of the Library	41
	3.5.2	Adding procedures to the Library	41
	3.5.3	Lowbound	41
3.5.	3.5.4	Range	41
3.6	Control Procedures		42
	3.6.1	STOP	42
	3.6.2	WAIT	42
Chapter 4:	ERROR INDICATIONS		
4.1	Errors During Translation of Program		43
	4.1.1	Run in Translation Mode (start at 8 or 12)	43
	4.1.2	Run in Report Mode (start at 10)	43
	4.1.2.1	Warning Messages	44
	4.1.2.2	Report Messages	44
	4.1.3	Error During Library Scan	45
	4.1.4	Undetected Errors	45
	4.1.5	Error Table at Translation Time	45
4.2	Errors During Loading of the Program		49
4.3	Errors During Running of the Program		50
	4.3.1	Undetected Errors	50
	4.3.2	Error Table at Run Time	51
Chapter 5:	OPERATION OF THE ALGOL SYSTEM		
5.1	General		53
5.2	Translation		53
5.3	Loading and Running		54
5.4	DUMP facility		55
5.5	Altering the built-in library		56
Appendix 1:	COMMON ERRORS MADE IN PROGRAM WRITING		
Appendix 2:	NOTES FOR USER'S OF ALGOL ON 920 COMPUTERS		
Appendix 3:	USE OF OPTIONAL PERIPHERALS		
Appendix 4:	USE OF EXTRA CORE STORE		
Appendix 5:	USE OF NON-STANDARD PERIPHERAL DEVICES		

Preface

The Algol system described in this manual can be operated on all computers of the 900 series.

The main body of the manual refers to a particular computer - the 903 - and one particular paper tape code - the 4100 (ISO) code. It can be read without change to refer to any other 900 series computer that uses the same paper tape code. Of course care must be taken that any machine code instructions contained in an Algol program obey the rules and conventions appropriate to the computer being used (see para. 3.4) and the peripherals attached to it.

Appendix 2 gives details of the changes to be made to the manual if it is to be used with a computer for which tapes are punched in 503 telecode, or one for which the tape readers are not designed to stop on a single character.

Chapter 1: SYSTEM SPECIFICATION

Minimum Configuration

The minimum configuration is a 903B computer equipped with 8192 words of core store, an input paper tape reader and an output punch.

Method of Operation

903 Algol produces an object program on paper tape which may subsequently be re-input for execution.

Efficiency

The translation speed is about 80 characters per second. Algol programs containing a normal proportion of floating point calculations will run at one fifth the speed of the corresponding machine code program.

Form of Distribution

903 Algol is distributed as a set of three binary tapes:-

(1) The translator in sumchecked binary.

(2) The interpreter and library combined in sumchecked binary.

(3) The library tape in sumchecked relocatable binary.

1.1 ELLIOTT 903 ALGOL REPRESENTATION

An alphabet of 68 characters is available for punching a program:-

A - Z	26
0 - 9	10
quote signs ' `	2
quote sign "	1
punctuation , . : ;	4
number signs + - * / $_{10}$ ↑	6
relations < = >	3
brackets ( ) [ ]	4
space	1
carriage return	1
line feed	1
erase	1
runout	1
halt	1
non Algol characters ½ $ % & £ ←	6
Total	68

1.1.1 Basic Symbols

Algol Symbol	903 Hardware Representation	Recommended mode of writing the symbol
a - z	A - Z	a - z
0 - 9	0 - 9	0 - 9
+ - /	+ - /	+ - /
×	*	*
÷	"DIV"	div
< = >	< = >	< = >
≤	"LE"	le
≥	"GE"	ge
≠	"NE"	ne
∩	"AND"	and
∪	"OR"	or
⊃	"IMPL"	impl
≡	"EQUIV"	equiv
¬	"NOT"	not
, . : ; $_{10}$	, . : ; $_{10}$	, . : ; $_{10}$
:=	:=	:=
( ) [ ]	( ) [ ]	( ) [ ]
␣	<space, newline, etc>	
begin end etc	"BEGIN" "END" etc	begin end etc
	"CODE" "ALGOL" etc	code algol etc

1.1.1.1 The Characters erase, runout and halt

The characters erase and runout may appear anywhere and are always ignored. The character halt is for stopping the computer at the end of a data tape or tape of Algol text, in order that the data or Algol text may be fed in as a series of separate tapes.

1. 1. 2 Punching Instructions

(1) A 903 Algol program may be punched on any one of several types of tape perforation equipment operating in the ISO code. Whatever equipment is used, the punched tape should either be verified or printed up, and the print-up produced from the tape checked against the original program manuscript.

On some equipment, new-line is punched as a single character, while other equipment uses separate carriage-return and line-feed characters. On the latter equipment N consecutive new-lines should be punched as:-

carriage return, N line feeds, blank.

(2) Capital and small letters are both punched as capitals.

(3) Underlining is represented by double quotation marks e. g. begin real a; is punched as "BEGIN" "REAL" A;

(4) Algol programs will be written either on a pre-printed form (the Elliott Algol Program Sheet) or on lined paper with some vertical lines added. The heavy vertical lines every six columns indicate where each line of print should start.

(5) Punch exactly what is necessary to produce a print-out like the written program, i. e. all blank lines, spaces, etc. Consecutive words, or consecutive underlined words, should be separated by a space. In general, the exact number of spaces, carriage return and line feed characters is not critical, but punching the correct number will improve the general appearance of the print-up. However, between the characters ' and ` the text must be punched exactly as written.

(6) Care must be taken to avoid confusion between characters, in particular between the figure '1' and the letter 'ℓ', and also between the figure '0' and the letter 'O'. These must be punched correctly and punch operators should familiarise themselves with the difference in print of these characters.

(7) There should be a runout of blank tape at the beginning and end of every tape punched.

(8) A wrong character may be cancelled by overpunching with 'erase'. This erase character does not contribute to the number of characters which may be punched in one line (See 12 below).

(9) Every semi-colon should be followed by three or more blanks. These blanks are a device to simplify editing if the program is subsequently found to need modification. Their omission is not an error, but is nevertheless undesirable.

(10) A halt code character must be punched at the end of every program tape, or at the end of every section of program tape if the program is punched in parts, and also at the end of every data tape.

(11) If it is impossible to punch one line of manuscript on one line of paper, a new line may be started anywhere except:-

Between : and = of :=

Between the string quotes ' and `

If it is necessary to split a word or number (and overpunching with erase and repeating the word or number on a new line is usually preferable) a hyphen must NOT be inserted.

(12) The symbol := is punched as : (colon) followed by = (equals). No characters other than blank or erase may be punched between these characters.

(13) There must not be more than 120 characters in a line.

1.1.3 The Use of Single Case.

In 903 Algol only one case of letters is available for the formation of identifiers. Programs which contain identifiers which are identical except for the use of upper-and-lower-case letters must be adapted in such a way that all identifiers which were originally distinct remain distinguishable when printed in a single case.

It is recommended that programmers should use only lower-case letters, although they will appear as upper-case on the printed version of the program. If it is necessary to change an identifier in a published program to avoid clashing, it is recommended

that one or more capital letters should be transcribed as pairs of the corresponding lower-case letter (e.g. A should appear as aa). If this produces a further clash of identifiers, each pair should be followed by a number sufficiently high to avoid all further clashing (e.g. aa1, aa2, etc.).

1.1.4 Notes to the Programmer.

(1) Algol programs may be written on a pre-printed form (Elliott Algol Program Sheet) or on lined paper (on which a series of vertical tabulation lines have been drawn).

(2) The programmer must clearly differentiate between the figure '1', the letter 'l', and also between the figure '0' and the letter 'O'. The use of continental 1 and 7 is strongly recommended for manuscripts.

(3) There must not be more than 120 characters on a line.

1.1.5 The Program

Every Algol program written for the 903 must be preceded by a title and the program must be followed by a semi-colon and a halt code in that order.

A title consists of a string of letters or digits of which the first character is a letter. It is terminated by a semi-colon. The first six characters of the title will be reproduced on the on-line teleprinter when the program is run*.

The title may also include spaces, carriage returns and line feeds, but these should not occur among the first six characters.

A title may not be the same as one of the Algol library procedure names (see Paragraph 3.4) and if it starts with the letter Q it must have U as its second letter.

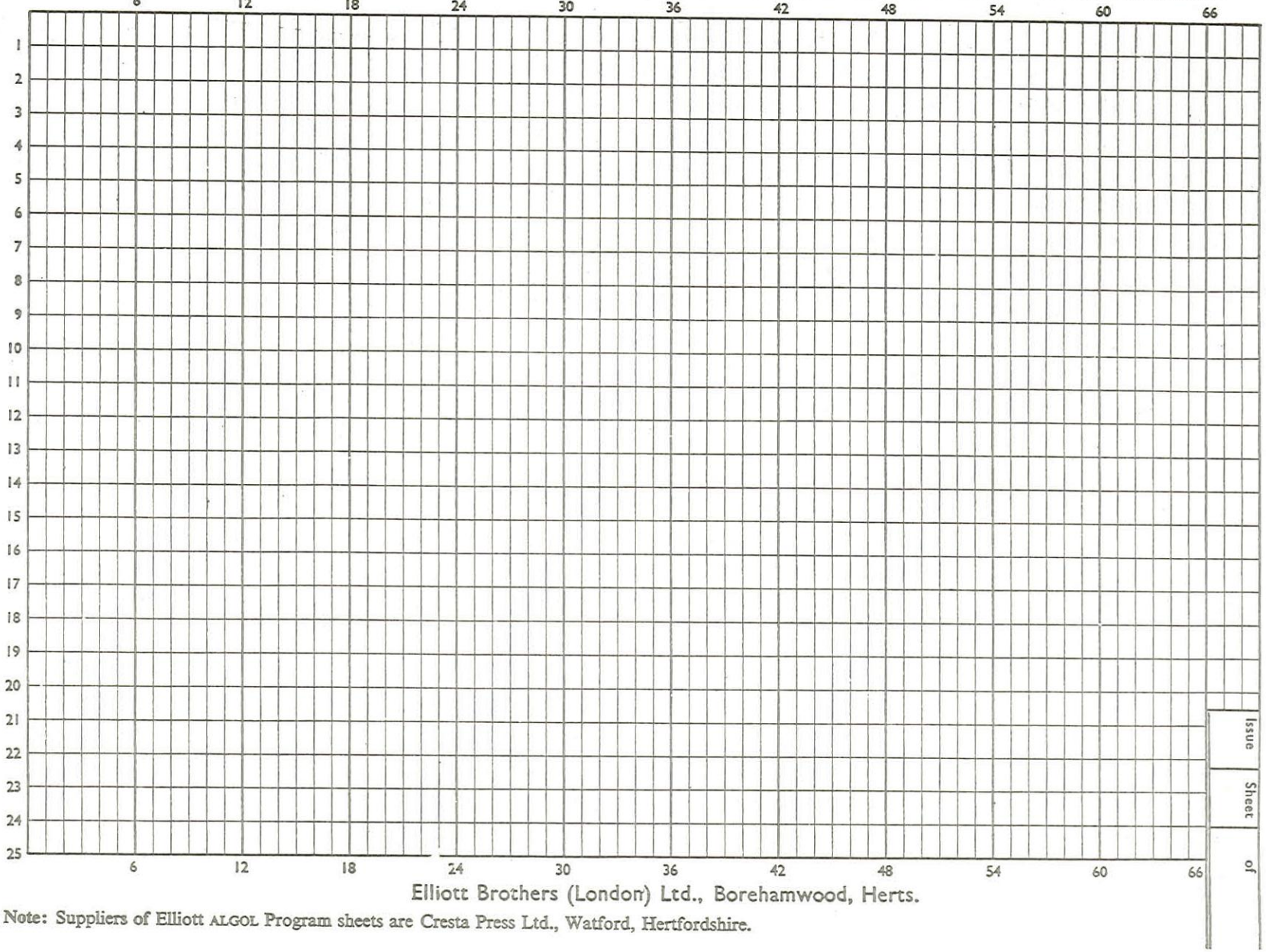
The title should be used to give enough information to identify the program and the programmer uniquely. Each installation is encouraged to establish its own standard practice for the writing of titles.

---

* Note: If an on-line teleprinter is not fitted, anything which would have been printed by it will be punched by the punch. Henceforth, the verb 'display' will be used for 'print out on the on-line teleprinter'.

ELLIOTT ALGOL PROGRAM

Title..... Subtitle.....



Elliott Brothers (London) Ltd., Borehamwood, Herts.

Note: Suppliers of Elliott ALGOL Program sheets are Cresta Press Ltd., Watford, Hertfordshire.

Since the title is not part of the program, a comment may not occur immediately after the semi-colon which ends the title.

1.1.6 The Use of Elliott Algol Program Sheets.

Elliott Algol Program Sheets (see Page 6) have been designed to enable the programmer to indicate to the punch operator the exact layout of his program. For this purpose, one and only one character should be written in each cell; a cell which does not contain a character will be treated as a space if it occurs in the middle of a line; after the last occupied cell of a line, a change to a new line will be punched. At the end of each program sheet three blank lines will be punched.

The cells are grouped in units of six by means of a firmer line. This is an aid to punch operators in counting the number of spaces required on a deeply indented line. It is recommended that identation should be a multiple of six.

For identification purposes, the title of the program should be written at the head of each sheet. If the sheet contains a recognisable subsection of the program, a subtitle may be used. These letters and subtitles at the head of the sheets will not be reproduced on the punched document. Therefore, the first sheet of the program must contain a copy of the title written in the text of the program, as well as at the head of the sheet.

1.1.7 Correction of Algol Programs.

Any errors in an Algol program, whether of a syntactic nature or in the actual formulation of the problem, must be corrected in the source language. There is no means of making corrections to the compiled program in the store of the computer.

To assist the programmer in editing his tapes, the halt code character is available. To insert a statement or group of statements into a program after a given statement, S say, punch a halt code on the tape immediately after the semi-colon that terminates S. On reading the halt code, the computer will come to a systems wait. Then translate the additional statements (the last one having a halt code after the terminating semi-colon), and finally continue translating the original program.

The halt code facility also permits a program to be punched in sections on separate tapes, each section having a halt code at the end.

According to the punching instructions, three or more blanks should be left after every semi-colon of the program. If this has been done, it is an easy matter to insert a halt code wherever necessary.

Alternatively, a 903 Algol program may be modified by the use of the 903 Edit program (Volume 2.3.2).

1.2 RESTRICTIONS AND PROGRAMMING NOTES.

This section describes a number of restrictions imposed on the full generality of Algol, and mentions restrictions which are part of Algol and whose effects are frequently overlooked.

1.2.1 Restrictions

1.2.1.1 The Declaration of Labels.

Any labels used to label a statement in the compound tail of a block must be declared in the head of that block:

```
switch ss:=label1, label 2, .... ;
```

The switch identifier is obligatory, even if it is not used in a statement of the program. Any identifier different from the other identifiers of the program may be used for a switch, but the use of a sequence of the letter 's' is recommended as standard practice, except where this would cause a clash with other identifiers.

Care must be exercised in declaring labels in blocks to which they are local. It is inadmissible to declare labels of an inner block in an embracing block. Note the difference between a compound statement and a block. The following is a scheme of correct label declarations:-

```
begin switch s:= L1, L2, L3;
    L1;
    L2: begin switch ss:= L3, L4, L5, L6
            L3:
            begin comment compound statement;
                L4:
            end compound statement;
            for .... do
              L5: begin
                  L6; goto L5;
                  end for statement;
            end block L2;
    L3:
end;
```

A label may not prefix a statement in a procedure body unless it is declared in the same procedure body. This may involve turning the statement which forms the procedure body into a block by attaching a begin and end and a switch declaration.

(1) Unsigned Integers as Labels.

Unsigned integers may not be used as labels, where they occur in a program, they should be turned into identifiers. The recommended practice is to precede each number by a sequence of one or more 'L's.

For example 23: might become L23 or LL23;

(2) Switches.

The elements of a switch list may only be labels. These labels must be prefixed to statements of the block in the head of which the switch declaration occurs. The occurrence of a label in a switch list serves as a declaration of that label and, therefore, no label can occur more than once in the switch declarations of any one block.

If in a published program these conditions are not satisfied, the switch declaration must be replaced by a portion of program which achieves the desired effect. It should seldom be necessary to do this, and no general rules are given.

If a goto statement uses a switch designator, and the subscript of the switch is negative, zero or greater than the number of labels in the switch list, then error message number four is displayed and no further statements of the program are obeyed. If this is expected to occur, the goto statement should be turned into a suitable conditional statement. For example:

```
goto ss [i];
```

where the switch list of ss has three elements, would become

```
if i>0 and i<4 then goto ss [i];
```

This has the effect specified in the Revised Algol Report.

1.2.1.2 Type of Arithmetic Expressions.

If i, j, k stand here for integer variables and a, b, c stand for real variables then:-

(1) i + j*a is real.

(2) i + (if j<k then 1 else b) is real since the else part produces a real result. Consequently the then part must also produce a real result, and the expression as a whole is real.

(3) i := j+a; j is converted to type real since a is real and then the sum (j+a) is converted to integer.

(4) i := a[j+b]; j is converted to real since b is real, but then (j+b) is converted to integer because it is a subscript.

(5) for list elements are converted to the same type as the controlled variable.

(6) Actual parameters are converted if necessary.

If the type of an arithmetic expression depends upon the evaluation of an expression or upon the type or value of an actual parameter then it is taken to be real, e.g. the result of exponentiation is always real even where both arguments are integer.

No accuracy is lost as a result of this.

Note: i↑3, i↑(3) and i↑(+3) are however of type integer as a special exception, but i ↑(-3) is real.

1.2.1.3 for statements.

The controlled variable in a for clause can only be a variable identifier. It may not be subscripted.

1.2.1.4 own declarations.

own declarations are not permitted.

1.2.1.5 Specification of Parameters.

All formal parameters of a procedure must be specified in the head of the procedure.

1.2.1.6 Recursive Procedures.

No recursive procedures are allowed.

1.2.1.7 Parameters.

An actual parameter called by name must be an identifier or a string, and must be of the same type as the corresponding formal parameter. The identifier may be the name of one of the following

simple variable
array
label
switch
procedure
string

If a formal parameter is called by name, the corresponding actual parameter must be the name of a variable of the same type as the formal parameter. Unsigned real and integer constants may however be substituted for real and integer variables, respectively.

1.2.1.8 Type Procedures.

A call of a type procedure can only occur in an expression; it cannot stand alone as a procedure statement.

1.2.1.9 Sequence of Declarations.

Declarations in a block head may occur in any order provided that no entity is referred to before the declaration of its identifier occurs in the text of the program.

1.2.1.10 Length of Identifiers.

Only the first six characters are significant in an identifier; subsequent characters are ignored.

1.2.1.11 Reserved Identifiers.

The identifiers checkr, checki, checkb and checks are reserved for special purposes and may not be used in any other way.

1.2.1.12 Limits

(1) The number of parameters of a procedure or dimensions in an array may not exceed 14.

(2) There can be up to 63 array names in an array segment.

(3) There can be up to 120 characters in an input line of text.

1.2.2 Programming Notes.

1.2.2.1 Range and Precision of Numbers

In all cases where an operation is defined as having a result of type integer, this result must be in the range

-131,072 to +131,071

If an integer exceeds these limits the program stops and error message number three is displayed.

In an arithmetic expression a constant is treated as a positive number with an associated sign. Thus in the example:

```
begin integer i;
          i:= -131072;
```

integer overflow results at translation time.

In all cases where an operation is defined as having a result of type real, this result must be in the range

-1×2^{63} to $(1-2^{-27}) \times 2^{63}$

i.e.

-9×10^{18} to $+9 \times 10^{18}$ approximately.

Zero is represented by 0×2^{0}.

If a real number exceeds this limit during the running of a program the computer stops and error message number nine is displayed.

During translation, a real constant that is too large is replaced by the largest real constant

$(1-2^{-27}) \times 2^{63}$

Where any operation (including reading and printing) is defined as having a result of type real, this result may be innaccurate by up to one part in 10^{8}. It is not recommended to expect an accuracy greater than seven significant decimal digits in testing convergence of a numerical process.

1.2.2.2 Sequence of Operations.

The order in which operations are performed within an arithmetic expression is undefined except insofar as it is determined by the rules of precedence:

(1) Exponentiation.

(2) Multiplication and Division.

(3) Addition and Subtraction.

Thus in

```
a:= b+c+d;
```

the order of evaluation of b, c and d is undefined so that, if one of them is a type procedure which affects the value of one of the others (a "sneaky" procedure), the value of a will be undefined. If it is important that b, c and d are evaluated in that order, they should be converted into expressions by the use of brackets:

```
a:= (b) + (c) + (d);
```

Similarly, if it is important that the two additions are performed in the order shown, the expression should be written:

```
a:= (b+c) + d;
```

If a, b and c are real and are of widely differing magnitudes, then (a+b) + c may not be equal to a+(b+c).

1.2.2.3 Boolean Expressions.

In the evaluation of boolean expression, every term is evaluated. Thus in the expression

```
a:= b or c or d;
```

even if b is found to be true, and thus the value of the expression determined, c and d are nevertheless still evaluated. Hence any side effects from c and d always occur.

Remarks analogous to those of 1.2.2.2 apply to the order of operations within a boolean expression.

1.2.2.4 Correspondence Between Formal and Actual Parameters

In most cases if the formal and actual parameters of a procedure are of different type, conversion of the actual parameter automatically takes place. However, in certain cases the correspondence between formal and actual parameters must be exact. There are as follows:-

(1) Array Parameters

If a formal parameter of a procedure is specified to be an array, the corresponding actual parameter must in all cases be of the same type and have the same number of dimensions.

(2) Parameters Called by Name

If a formal parameter to a procedure is called by name, the corresponding actual parameter must be an (unsubscripted) identifier or string, or a constant. The actual parameter must correspond exactly in type and kind to the formal parameter. If the call by name formal parameter has a value assigned to it within the procedure the actual parameter must be a simple variable.

(3) Procedure Parameters

When a procedure has a parameter which is specified as a procedure, the first occurrence of the formal parameter procedure within the procedure body shows the number and type of its parameters.

All calls of the parameter procedure thereafter, within the procedure body, must have parameters which correspond exactly in number and type.

For example:

```
procedure f (x); procedure x;
    Begin real a,c; integer b,d;
        x(a,b);
        x(c,d); comment NOT x(a,c) because the type of c
                is incorrect, or x(a) because number of
                parameters is wrong;
    end f;
```

If in the example above there had been x(a,c); then error message one would have been given at translation.

In any call of f, the actual procedure used to replace x must have parameters which correspond in number and type to those used in the body of f. The parameters of this procedure must be called by name. If these rules are not obeyed an error will be given at run time.

If the formal procedure x was a type procedure, this must be declared in the heading of f, e.g.:

```
procedure f(x); real procedure x;
    a:= x(a);
```

The restriction on call by value parameters may be evaded by use of a local variable in the parameter procedure.

```
e.g. procedure A(X); value X; integer X;
     begin  . . . .
     may be replaced by

     procedure A(Y); integer Y;
     begin integer X;
         X:=Y;
         . . .
         . . .
```

It should be noted that with the exception of instring and outstring the standard procedures of 903 Algol have parameters called by value.

1.2.2.5 goto if

The destination of a goto statement must not be conditional, e.g. the construction

```
goto if A then L1 else L2;
```

is not allowed. It may be replaced by

```
if A then goto L1 else goto L2.
```

Chapter 2: INPUT AND OUTPUT

2.1 INTRODUCTION.

The Revised Algol Report does not specify the mode in which statements causing data to be input and results output are to be written. The mechanism of read and print statements used in other Elliott Algol implementations has been adapted to 903 Algol. Programmers who prefer to use read and print procedures may readily declare such procedures with or without parameters and, if with parameters, with the number and type of parameters with which they are familiar.

2.1.1 Print and Read Statements.

Two types of statement are introduced, the print statement and the read statement. The syntax of these statements is very simple, since they consist only of the words print and read, followed by a list of operands, the items of the list being separated by commas, e.g.

```
read x, y, b[j];
print x↑2, x*y, z, cos(x/q)-b[j];
```

The effect of a read statement is to cause one or more numbers to be read from the paper tape reader and assigned in the sequence read to the variables specified in the list. The effect of a print statement is to punch the values of the arithmetic expressions occurring in the list.

2.1.2 Structure of Read and Print Lists.

The elements of a read or print list may be arithmetic variables or procedures. A print list may also contain arithmetic expressions and strings. These elements are scanned in the sequence in which they are written, with the following effect.

ITEM	EFFECT IN A READ LIST	EFFECT IN A PRINT LIST
arithmetic variable (can be subscripted)	A number is read from the paper tape reader and its value is assigned to the named variable	The value of the variable is output on the current device
number	NOT PERMITTED	As for arithmetic variable
expression	NOT PERMITTED	As for arithmetic variable
non type procedure	The procedure body is executed	The procedure body is executed
arithmetic procedure (inside the body of the procedure)	(The procedure name on its own) As for an arithmetic variable	NOT PERMITTED
arithmetic procedure (not inside its own body)	NOT PERMITTED	The procedure body is executed and the value of the procedure is printed on the current output device.
array name	NOT PERMITTED	NOT PERMITTED
Boolean variable	NOT PERMITTED	NOT PERMITTED
Boolean procedure	NOT PERMITTED	NOT PERMITTED
string	NOT PERMITTED	The string is printed on the current output device.

2.1.3 Input Data Tape.

Numbers to be read by the computer should conform to the ALGOL definition of a number, and each number must be followed by some character other than a digit, a decimal point or subscript ten. This terminal character may be followed by any sequence of characters excluding the digits 0 to 9 and the characters . $_{10}$ + - . The sequence may be of any length, its only function is to separate the numbers on the data tape, and it is otherwise completely ignored. Note that spaces may not be used in the middle of numbers.

Blank tapes and erase are ignored under all circumstances. Halt causes the computer to wait.

The character ' is not permitted on a data tape, except in connection with the "instring" facility.

The example data tape

ITEM 176329
197 FT 7 INS
G=1.276$_{10}$-11
7.392$_{10}$-4
710623 1.49700

is read as a succession of numbers

176329	(too large for integer)
197	
7	
1.276$_{10}$-11	
7.392$_{10}$-4	
710623	(too large for integer)
1.49700	

2.1.4 Presumed Settings.

The print and read statements of the simple type explained above are executed under the control of the presumed settings.

Methods of changing the settings and obtaining varied and more sophisticated effects of format control are described in 2.2 of this chapter.

Numbers printed under the control of the presumed settings appear on separate lines. In the case of integers, up to six digits are printed with leading zeros replaced by spaces, and, in the case of a negative number, the sign is floated i.e. moved along so that it immediately precedes the most significant digit. In the case of real numbers of magnitude less than 100,000,000 eight digits are printed preceded if necessary by a minus sign. Larger numbers are printed with an exponent part and only four significant digits.

Examples of printing under presumed settings:

Integers	Real Numbers
9	-9.000000
-127	127.3061
-125000	-2396123
	.0000000
	-1.36₩+12

2.1.5 Output of Text.

In order to output headings and other messages, the characters which comprise the message should be enclosed within string quotes and included as an item of a print statement; these items are output in the sequence in which they appear.

Examples

```
print ' height weight speed ` ;
print f, ' ft `, i, ' ins ` ;
```

To output carriage returns, line feeds, spaces, run-out or the symbols ' and ` , an inner string should be used which consists of certain special interpreted characters enclosed by a further pair of string quotes ( ' and `).

These characters may appear inside single string quotes amongst text which is itself enclosed in string quotes. The meanings of the special characters are given in the following table:

ℓ	new line
s	space
r	runout
q	quote '
u	unquote `
h	halt

If any of these characters is followed by an unsigned integer, the effect is the same as writing the character the specified number of times.

To facilitate the production of special effects the character b followed by an unsigned integer may be used in an inner string. The effect is to output the 8-bit character with the indicated binary value. (In calculating the desired value the parity bit (value 128) must be explicitly included if it is required). There is no automatic repetition facility in this case.

Spaces and new lines may also occur within a single pair of string quotes, and they are reproduced in the same way as other characters; care must be taken to ensure that only those characters required are quoted.

Examples

```
print"ℓ4` chapter`, n,"ℓ2s5";
print"ℓ`height's12`weight's ℓ2` speed ;
for i:=1 step 1 until 60 do print"b0";
print"ℓ`height          weight          speed` ;
```

Text is not preceded by a change to a new line and, therefore, is printed on the same line as the last number. A change to a new line may be specified as part of the text.

The third example has the same effect as print "r60"; and the second example has the same effect as the fourth.

2.2 SETTING PROCEDURES

The presumed settings are adequate for inexperienced programmers, for program testing and low volume output. In many applications, however, the programmer will wish to use all available input and output devices and to exercise control of output format. For these purposes a number of procedures (setting procedures) are provided.

A setting procedure statement normally occurs in the list of a print or read statement, separated in the normal way by commas from the other operands. All numbers and text in a list subsequent to a setting procedure will come under the control of the settings specified. Any number occurring previously in the list or in the lists of subsequent print statements is unaffected. Thus, operations in a list previous to a setting procedure statement are not affected by it.

A setting procedure statement may also occur as a separate statement of the program. In this case, once it is obeyed, it influences all subsequent printing until altered by a contradictory setting procedure statement. Such alteration may occur locally inside a print statement or globally as a separate statement of the program.

Where a parameter of a setting procedure is of type integer, the actual parameter may be an expression, allowing the format of results to be determined dynamically.

2.2.1 Device Setting Procedures.

In order to use input and output devices other than the paper tape punch and reader, it is only necessary to specify the device required in the list, prior to the operands concerned. This is done by calling the standard procedures reader(I) and punch (J). The available devices are:

reader (1)	The paper tape reader
reader (3)	The on-line teleprinter
punch (1)	The paper tape punch
punch (3)	The on-line teleprinter

For other devices see Appendix 3.

Example

```
print x, punch(3), "'l' feed tape", N;
```

will cause the value of x to be printed on the punch (the presumed setting), and the message 'feed tape' followed by the value of n to be displayed on the teleprinter for the attention of the operator.

2.2.2 Prefix Setting Procedures.

The procedure sameline is used to print several numbers on one line. It suppresses the new line sequence which is normally output before each number.

The statement

print t, 'tons', sameline, c, 'cwt', l, 'lb' ;

causes the value of t to be printed on a new line (the presumed setting), followed by the rest of the numbers and text on the same line e. g.

17 TONS 12 CWT 19 LB

If all the numbers to be printed are to be preceded by the same character or characters, the procedure

prefix ('. . . .')

should be used. This causes the text displayed between 'and' to be printed (in the place of new line) before every number. The procedures prefix and sameline are mutually exclusive.

For example, to print five numbers on a line separated by a comma and two spaces, write the statement:

print a, prefix(', ' ss"), b, c, d, e;

2.2.3 Format Setting Procedures.

The following procedures are available to change the number of digits printed and the style in which real numbers are output.

Procedure	Effect
digits (n)	Print integers with n digits ($1 \leq n \leq 12$)
freepoint(n)	Print real numbers with n digits ($1 \leq n \leq 8$) and a decimal point in the appropriate place.
scaled(n)	Print real numbers with 1 digit before the decimal point and (n- 1)digits ($2 \leq n \leq 8$) after it and an exponent indicating the power of ten by which the printed number is to be multiplied.
aligned(m, n)	Print real numbers with m digits before the decimal point and n digits after it ($1 \leq m+n \leq 15$)

Numbers are preceded by a space or minus sign as appropriate. The formats freepoint, scaled and aligned are mutually exclusive.

If a number is too large to be printed under the current format setting the effect described in Paragraph 2.2.4 below occurs.

Examples

```
print i, digits(4), j, k, l;
```

could produce the output

```
  -213
-1024
  362
   -7
```

```
print x, prefix(',' s2"), scaled(4), x, aligned(3, 4), x, freepoint(4), x;
```

would cause the following sample output if it were to be obeyed repeatedly for different values of X

```
-1.234568,   -1.235␣+00,     -1.2346,   -1.235
 123.4568,    1.235␣+02,    123.4568,    123.5
 .0012346,    1.235␣-03,      0.0012,    .0012
```

2.2.4 Alarm Printing.

If an attempt is made to print a number which is too large for the style of printing called for, e.g. to print 289.2 in the aligned (2, 4) format, or print 12347 in the digits(4) format, then "alarm printing" occurs in such a way that the layout of the printed page is undisturbed. Provided that the total number, N, of characters called for is greater than six, the number is printed in scaled (N-6) format; if $N \leq 6$, the letter H and a halt code are printed preceded by (N-1) spaces.

An attempt to print a real number which is not in standard floating point form, causes error message number seven to be displayed and *** to be output on the current device.

2.2.5 Parameters out of Range.

If the parameters of a format setting procedure are outside their permitted range, a return to presumed settings occurs as follows

Procedure Called	Permitted Range of Parameters	Presumed Setting
digits(e)	$1 \leq e \leq 12$	digits(6)
freepoint	$1 \leq e \leq 8$	freepoint(7)
scaled(e)	$2 \leq e \leq 8$	freepoint(7)
aligned(e,f)	$1 \leq e+f \leq 15$	freepoint(7)
reader(e)	e=1 or 3	reader(1)
punch(e)	e=1, 3 or 4	punch(1)

2.2.6 Input and Output of Strings.

Strings may be read in as data and printed out again by using the procedures

instring(a,m)
outstring(a,m)

where a is a one-dimensional integer array and m is an integer variable. (Neither a subscripted variable nor a constant may be substituted for m, which is called by name and to which a value is assigned by the procedures.)

The effect of instring is to search input characters on the current device , searching for a ´. If a numeric character is encountered during this search, error message number fourteen is displayed. The string that starts with the ´ is stored, packed three characters to a word in locations a[m], a[m+1].... When the ` which brackets with the original ´ is encountered it is stored, input stops, and the variable m is set to the index of the next available element of a. Thus, on repeated execution of the procedure, m is automatically set in such a way that strings cannot overwrite each other.

The devices used for input by instring or output by outstring are those specified 'globally', that is by the last call of setting procedures reader(I) or punch(J) occurring outside a read or print list. This applies even if the procedures instring or outstring are called from within a read or print list, which differs from the effect on the 803, 503 and 4100. Setting procedures inside a print list have no effect on instring or outstring.

The effect of outstring is to output a string which has previously been input by the instring facility. The integer variable m indicates the index of the first of the elements of a which contain the string, and the procedure will set m to the index of the first element containing the next string. Thus, on repeated execution of the procedure, successive strings stored in the array are output in succession. Inner strings are interpreted in the same way as that specified for strings occurring in print statements.

If the specified array is not one-dimensional and integer, or if the initial or final values of the subscripting variable are out of range, then error message number 23 is displayed. If an illegal character is found in an inner string, error number 6 is displayed when outstring is obeyed.

The array a should be sufficiently large to contain all the strings read. A string of n characters, where n includes the outermost string quote signs and every character between them, will occupy (n+2) div 3 locations.

2.2.7 Procedures in Read and Print Lists

The body of a procedure occurring in a print list may contain read and print lists. On reverting to the original list the current format is the one in force at the end of the last read or print executed within the procedure body. This differs from the effect on the 803, 503 and 4100.

Chapter 3: STANDARD PROCEDURES

3.1 INTRODUCTION

The procedures described in this chapter are available without specific declaration. They may be thought of as declared in an outer block in which the program being run is embedded.

3.2 ALGOL STANDARD FUNCTIONS

The function procedures listed in 3.2.4 of the Algol 60 Report are available without specific declaration. They all have their argument (which is of type real) called by value.

3.2.1 real Procedures

abs(E)	the modulus of the value of the expression E
sqrt(E)	the square root of the value of E ($0 \leq E$)
sin(E)	the sine of the value of E ($E < 2^{18}$)
cos(E)	the cosine of the value of E ($E < 2^{18}$)
arctan(E)	the principal value, in radians, of the arctangent of the value of E
ln(E)	the natural logarithm of the value of E ($0<E$)
exp(E)	e^E

The argument of 'cos' and 'sin' is in radians, the limit on its value is imposed to ensure that it has enough significant figures, after multiples of 2π have been removed, for the evaluation of the functions to be meaningful.

3.2.2 integer procedure sign (E)

This procedure takes the values:

+1 if E>0
+0 if E=0
-1 if E<0

3.2.3 entier (E)

The transfer function entier is available without specific declaration. It is an integer function with a real argument called by value. Its value is the largest integer not greater than the value of E. It should be used with care because of the finite accuracy of real numbers:

e.g. entier (1.9999999) = 1
entier (-1.000000001) = 2

3.3 CHECKING FUNCTIONS

The standard checking functions provide the programmer with a means of optional printout of intermediate results during the course of a calculation.

There are three such functions

checkr for real argument
checki for integer argument
checkb for Boolean argument

and they are written in the program as functions of a single parameter which is called by value.

e.g. a:=checkr(b+2*c)+1.5;

If the program is translated in the normal mode the checking functions are ignored - that is to say the statement is treated exactly as if it were

a:=(b+2*c) + 1.5;

If the program is translated in checking mode then an extra order is compiled so that the value of the argument is output. Thus b+2*c would be printed.

In the case of Boolean checking the output is one of the words true or false. In all cases check output is preceded by newline, asterisk.

The checking functions may be nested, thus it is possible to write

a:= b+checkr(m[checki(i+3)]);

in which case the value of (i+3) would be output before the value of m[i+3].

If two or more non-nested calls of the checking functions occur in the same expression, then the order of evaluation of the terms comprising the expression may differ according as it is translated with or without checks. (See Paragraphs 1.2.2.2 and 1.2.2.3.

If checkr is used with an integer argument or checki with a real argument, then type conversion will occur when the program is translated with checks, but not when it is translated normally. This can lead to several unintended and unexpected effects.

In addition to the three checking functions there is the checking procedure

checks

whose parameter is a string not containing a semi-colon or closing round bracket ')'. It can conveniently be used to provide a form of trace of the progress of a calculation, or to identify the output of the checking functions. A typical call of this procedure would be

checks(`stage 1 complete`);

3.4 MACHINE CODE.

Certain operations can be performed much faster when expressed in 903 machine code than when expressed in Algol. This is because the Algol translator must cater for all possibilities, but the programmer writing in machine code need only consider his special case.

It is particularly effective in matrix work.

All machine code must be in the form of special procedures and communication with Algol is via the parameters of these procedures.

The procedure is written in two parts, an Algol declaration and a code body. The declaration is included in the Algol text. The procedure body is written in 903 SIR code and translated separately by the SIR assembler (Volume 2. 1. 1).

3. 4. 1 Declaration in Algol text.

The declaration of a code procedure is similar to the head of a normal procedure declaration, but it is not followed by a procedure body and it must be enclosed within the special brackets

code and algol;

For example

```
code
    integer procedure SUM(A, B);
    value A; integer A, B;
algol;
```

Note the semi-colon after algol. Subsequent calls are as for ordinary procedures although they are translated differently.

The standard procedures on the library tape behave as though a declaration in this form occurred in a block surrounding the program.

3. 4. 2 As written in SIR

The machine code is written in SIR code and converted by the SIR assembler to a relocatable binary tape which can be linked to the Algol in alternative ways; either it is added as an extension to the library procedures or it can be fed in just before execution as a separate tape.

It may be necessary to refer to a number of fixed locations which hold a 16 bit address. These locations are:

Absolute Address	Contents	Explanation
132	QACODL	Algol Constants Object Data Load. This points to the start of an area containing constants, switches and labels. A goto switch expression or label parameter requires reference to 132.
137	EP	Entry Pointer. This points to a group of locations holding the status of the current block. A goto switch expression or label parameter requires reference to 137.
138	FP	Formal Pointer. Most machine coded procedures need to refer to 138 (see below).
140	PBA	Primitive Base Address. This points to a list of basic interpreter subroutines. A machine coded procedure needs to refer to 140 only if it makes use of interpreter subroutines.
180	W	Workspace Pointer. Reference to 180 is only needed when interpreter subroutines are used.

3. 4. 2. 1 Scalar Parameters.

The table below shows how the contents of 138, the Formal Pointer FP, are used to access scalar parameters. Reference to a parameter numbered n is via location 3n+(content of FP) and sometimes via the next two locations as well †:-

Type of Parameter	mode of call	Contents of		
		3n+FP	3n+FP+1	3n+FP+2
integer or Boolean	by name	address	undefined	undefined
	by value	value	undefined	undefined
real	by name	address*	See Note 1	undefined
	by value	<------mantissa------->		exponent

A boolean variable has the same form as an integer variable and the two possible values are

true is held as +1
false is held as zero

3. 4. 2. 2 Result of a Type Procedure.

The Formal Pointer points to the result.

Type of Result	Contents of		
	FP	FP+1	FP+2
non-type procedure	(no result)		
integer or boolean procedure	result	undefined	undefined
real procedure	<------mantissa------->		exponent

Example:-

(1) A procedure declaration in Algol

```
integer procedure SUM(A, B);
value A; integer A, B;
SUM:= A+B;
```

---

† The value of a type procedure is assigned to parameter 0.

* But the sign bit is set to 1.

Note 1 This location holds an indicator word. If the word is positive the real number is stored in packed format, if negative it is in unpacked format (see 3. 4. 9).

(2) The same procedure declared, if its body is to be in code.

```
code
      integer procedure SUM(A, B);
      value A; integer A, B;
algol;
```

(3) The code body, to be written separately from the Algol text.

```
*+0
[ SUM ]
SUM   /14        2        (TWO PARAMETERS)
      +0
      0          138
      /0         6        (LOAD SECOND PARAMETER)
      /4         0        (CALLED BY NAME)
      0          138      (ADD FIRST PARAMETER)
      /1         3        (CALLED BY VALUE)
      /5         0        (PUT RESULT ON STACK)

      0          SUM+1
      /8         1
%
```

For a typical entry to this subroutine the content of location 138 might be +5500. Then locations 5500 onwards might contain:

```
5500   + X1   (Undefined, result space for function)
       + X2
       + X3

5503   + 10   (Value of A)
       + X4   (Undefined)
       + X5

5506   + 4600 (Address of B)
```

and location 4600 might hold +20, the value of B.

3.4.3 Rules for SIR Blocks.

The following rules apply to machine code procedures written in SIR.

(1) Each procedure must have *+0 before the first global list. No other options can be used anywhere on the tape.

(2) There must be one global label and this is the name under which the procedure is known. The procedure must be one and only one block.

(3) The procedure name must differ from that of any other procedure on the library tape. Furthermore if it begins with Q its second character must be the letter U.

A list of names present on the standard library tape is given in Paragraph 3.5 below.

(4) The global label must appear on the left in the first location of the procedure which must contain the word /14 n where there are n parameters to the procedure.

(5) The rest of the machine code follows as an ordinary subroutine expecting to have its link stored in the second location and to be entered at the third location.

(6) The block is terminated by the % character.

(7) The following SIR facilities are available.

7.1 Constants
7.1.1 Integer e.g. -79
7.1.2 Fixed Point fraction e.g. +.317
7.1.3 Octal e.g. &770123
7.1.4 Alphanumeric e.g. £A23
7.2 Addresses
7.2.1 Absolute e.g. 132
7.2.2 Block relative e.g. 5;
7.2.3 SIR relative e.g. ;-3
7.2.4 Identified e.g. LIST+95
7.2.5 Literal i.e. any constant or = followed by an instruction with an absolute address.
7.3 Skips e.g. >+5
7.4 Comments e.g. (this is a comment)

Other SIR facilities are not available.

(8) The effect of altering the contents of an absolute location, other than locations 180 to 203 (see below), is undefined. In particular the contents of locations 132, 137, 138 and 140, which are frequently referred to within CODE bodies, must not be altered.

(9) The length of each machine code block must not exceed 900 words.

3.4.4 Array parameters.

When an array parameter is specified for a code (SIR) procedure it is necessary to hand over details of the array and store layout. Three distinct areas of store are specified; a pair of pointer locations, the array map and the array itself. The array map is a block of store holding information on the size of array, subscripts and bounds. The actual array is a continuous block of store, with either one word for each element of an integer or boolean array, or two words for each element of a real array.

The following examples are used to illustrate the description:

integer array A[1:100];

array B[1:3, 2:3];

The elements of a multi-dimensioned array are stored in order such that the first subscript varies most rapidly; e.g. array B is in order:

B[1,2], B[2,2], B[3,2], B[1,3], B[2,3], B[3,3].

If the nth parameter is an array, on entry to the code procedure location 3n + (content of FP) will always hold an address. This address will point to the first of a pair of 'pointers'. The first pointer location holds the address of the actual array block in store. If the array is real then bit 18 of the first pointer location will be set to one (i.e. the location will be negative). The second pointer location holds the address of the array map.

The array map is 2d +2 words long, where d is the number of dimensions in the array. It has the following form:

General form	Array A	Array B
Number of dimensions (d)	+1	+2
Total size	+100	+12
Offset	-1	-14
Lowbound 1 (L_1)	+1	+1
Constant 1 (C_1)		+6
Lowbound 2 (L_2)		+2
Constant 2 (C_2)		
Lowbound 3 (L_3)		
etc.		

Total size is the total number of words in the actual array. Offset, C_1, C_2, etc. are constants used in array subscripting. L_1, L_2, L_3 etc. are the value of the lower bounds for the first, second, third subscripts. This information in the array map is not necessary if the elements of the array are to be accessed in sequence, without checks on subscripts.

The store address of element X[i, j, k, ...] is calculated from:

(Address of array) + Offset +

$f * i + C_1 * j + C_2 * k + \ldots\ldots\ldots$

Where F = 1 for integer and boolean arrays and

F = 2 for real arrays

If the array parameter is called by value in the procedure specification there is no difference in the information handed to the code procedure. If the effect of a call by value is required the code procedure itself should make a separate copy of the array, and operate on the copy.

Examples

Code procedure PRA has two parameters; the first is an integer array, the second is a real array.

On entry to PRA, the following values might be set up:

Location	Contents	Meaning
138	+6000	FP
6003	+4800	First parameter (address)
6006	+4790	Second parameter (address)
4800	+5004	Address of actual array A
4801	+5000	Address of array map for A
4790	/0 5110	Address of actual array B (real)
4791	+5104	Address of array map for B
5004	+12345	First element of A
5110	0.5	First element of B
5111	+63	(0.25 in 2 word floating point)

Program

To access first element of B:

```
 0  138
/0    6
/4    0
 5    ADDR
 0    ADDR
/4    0
 5    W
/4    1
 5    W+1
```

To get Total size of A:

```
 0  138
/0    3
/0    1
/4    1
```

(Total size in A-register)

(value of first element in W, W+1)

(To unpack to 3 word format, see 3.4.9.1)

To get address of element B[I, J];

```
 0  138
/0    6
/0    1      (address of map)
 4    J
/12   4      (J * C1)
 14   17     (Integer multiplication)
 1    I
 1    I      (Add 2 * I)
/1    2      (Add Offset)
 9    FAIL   (subscript out of range)
/2    1      (total size - relative address)
 9    FAIL
 7    FAIL
/2    1      (Restore relative address)
 0  138
/0    6
/1    0      (Add address of array)
 5    ADDR
```

The 5 instructions from 9 FAIL to the second /2 1 inclusive can be omitted if subscript checks are not required.

3.4.6 Label Parameters

Exit from a machine code procedure to a label parameter is illustrated by the following program. An absolute address in 3n + FP is converted into a relative address, and function bits added to form an instruction to be interpreted as "GOTO" by the Interpreter. The address of the location holding this instruction is placed in the stack so that control is transferred to the GOTO function when the normal exit is taken from the code procedure.

To exit from the procedure PROC to a label which is the second parameter, the following code may be used:

```
        4 132
        0 138
       /2   6          (/2  n*3 for the nth parameter)
        6  +8191
L1      1  =10 0
        5  LIVE         (Store GOTO instruction)
        4  ADLIVE
        0 137
       /5   1
        0  PROC+1       (Normal Exit)
       /8   1
ADLIVE   0 LIVE
  LIVE  +0
```

3.4.6 Switch Parameters

Exit from a machine code procedure to a switch parameter is illustrated by the following program.

The procedure PROC (in the example in 3.4.5) has a switch as its third parameter. N holds the required switch subscript value. The first 8 instructions only check the subscript and they may be omitted:

```
        4   N
        7   FAIL
        9   FAIL  (SUBSCRIPT TOO SMALL)
        0  138
       /0   6
        4   N
       /2   0
        9   FAIL  (SUBSCRIPT TOO LARGE)
        4  132
        0  138
       /2   9
        1   N
        1   N
        1   -1
        8   L1    (see example in 3.4.5)
```

Should be 109

AJ HERBERT 6/3/2012

3.4.7 String Parameters.

Where the nth parameter is a string (3n + content of FP) will hold an address, pointing to the first word of the string. A string is stored in SIR internal code with its opening and closing string quotes, left to right, left justified and space filled at the right. Thus 'CAT' is held as:

Word 1	£'CA	(&074341 octal)
Word 2	£T'Ⓢ	(&644000 octal)

3.4.8 Use of Interpreter Subroutines.

Procedures written in machine code may use subroutines in the interpreter. The subroutines are numbered and those which may be used are as follows:-

28	R1 := R1 ↑ I2
31	R1 := R1 + R2
33	R1 := R1 - R2
35	R1 := R1 × R2
36	R1 := I1/I2
37	R1 := R1/R2
38	I1 := I1↑I2
39	R1 := I1↑I2
40	R1 := R1↑R2

The user must place in location 180 the address of the first location of his workspace. For example, suppose that there are two <u>real</u> numbers, labelled R1 and R2, and that R1 is to be replaced by R1-R2 then the following program does this:

```
                .
                .
                4    POINT
                5    180
                0    140
                /0   33
                /11  0
                /8   1
                 .
                 .
                 .
POINT           0    R1

R1              >+3
R2              >+3
```

For integer operands, I1 and I2, the labels must be three locations apart as for R1 and R2.

3.4.9 Real Parameters

All real arithmetic in code procedures uses the unpacked (3 word) format, with the fraction mantissa in the first two words and the exponent in integer form in the third word.

However the elements of a real array and real parameters called by name (with the exception in 3.4.9.3) are stored in the packed (2 word) format (described in Volume 2.2.8 Chapter 1.2.2).

3.4.9.1 Unpacking

The following program will unpack a real number, given the address of the number in ADD, and leaving the result in location W etc.

```
0     ADD
/4    0
5     W
/4    1
6     &377600
5     W+1
/4    1
14    11
14    8181
5     W+2
```

3.4.9.2 Packing

The following program will pack the real number stored in locations 183, 184, 185 into the two words specified by an address which must be set by the user into location 203. This address may point to workspace in the code procedure, or an array element or real variable.

```
0     174
/11     0
/8    24
```

Locations 183, 184, and 185 are left undefined. All interpreter subroutines affect 183, 184 and 185, in particular the subroutines number 28, 31, 33, 35, 36, 37 and 40 (in 3.4.8) leave their real results in these locations, as a side effect.

3.4.9.3 real Parameters called by name

Real parameters called by name are stored in packed form unless the actual parameter to the machine code procedure is itself a formal parameter called by value. The location 3n + (contents of FP) holds the address of the actual value with the sign bit set to one. The location 3n + 1 + (contents of FP) is positive if the value is packed, and negative if the value is unpacked.

3.4.9.4 Integer to real conversion

Conversion of an integer value to an unpacked real value is achieved by means of the standardising subroutine, as in the following example:

```
    4     I      (Store integer in floating-point form)
    5    183
    4    +0
    5    184
    4    +17    (Integer is stored *2↑17)
    5    185
    0    150
  /11      0    (Convert to standard form)
   /8      1
```

This leaves the real value in standardised floating-point (unpacked format) in locations 183, 184 and 185.

3.4.9.5 Real to integer conversion

Conversion of an unpacked real value to the corresponding integer value may be achieved by the following subroutine entry:

```
    0    157
  /11      0
    8    158
```

This code will convert the real value in 183, 184 and 185 into an integer value. The integer value is given in the A-register and location 183 on exit from the subroutine. Conversion gives the value: entier (X+0.5) where X is the real value, (i.e. rounding to the nearest integral value). If the result is outside the range -131072 to +131071, error number 3 will be output, and the program will stop.

3.4.10 Use of interrupts in SIR code procedures

Algol programs are run on level 1. The user may write his own code routine to drop to level 4, the base level. Levels 2 and 3 are then available for use by code routines. The S-registers for levels 2 and 3 must be set to appropriate values, normally by means of a SIR code procedure within Algol. Once the appropriate values have been set up, interrupts may be used. The interrupt routine may be quite independent of the Algol, or it may be used to control the Algol program. If completely independent of the Algol program,

it may be incorporated in the built in library and the level 2 and 3 S-registers set up by making a keyboard entry to the address given in the store map. The example below shows a procedure used to control an Algol program loop. The procedure IFL2 is a boolean procedure entered repeatedly by an Algol program. If the level 2 Manual interrupt button has been used it takes the value true, otherwise it is false.

```
          *0
         [IFL2]
IFL2     /14   0
         +0
          4   L2A
          5    2          (Set level 2 S-register)
          4   F
          0  138
         /5    0
          4   +0
          5   F
          0   IFL2+1
         /8    1
F        +0
L2A       0   L2          (Address of level 2 routine)
L2        5   A
          3   Q
          4   +1
          5   F
          0   Q           (See note below)
         14    1
          4   A
         15  7168
          8   L2
A        +0
Q        +0
          %
```

Once IFL2 has been used for the first time, the level 2 Manual button may be pressed. This causes an interrupt, and the program sets F:=1 (Note that in this example it is not actually necessary to store Q, since the interrupt program does not affect it). The next time IFL2 is used, it will take the value true. This might cause the Algol program to break out of the main loop, to input a value from the on-line teleprinter, for example:

```
"CODE" "BOOLEAN" "PROCEDURE" IFL2;"ALGOL";
LOOP: CALC(A,B,X,Y); "PRINT"X;
"IF" IFL2 "OR" X<Y "THEN" "GOTO" ENDL "ELSE" "GOTO" LOOP;
ENDL: "READ" READER(3),Y;
```

3.4.11 Loading Machine Coded Procedures

3.4.11.1 As an extension to the library

Copy the existing library tape up to but not including the special terminating character (binary 01110000). Add a copy of the relocatable binary version of the machine code procedure. See Paragraph 3.5.1 below for the structure of the library tape.

The procedure will, therefore, be copied on to the end of any Algol program which required it. The declaration of the code procedure must be in the outermost block of the Algol program if this method is used.

This would be sensible for a well-tested machine code procedure.

3.4.11.2 At Run Time

Several possibilities exist here depending on whether the rest of the library is in the store or not.

(1) Library in the store already and part of it needed. Read in the Algol by starting at 8 and then the machine code by starting at 11.

(2) Library in the store already but not needed. Read in the Algol by starting at 13 and then the machine code by starting at 11.

(3) Library not in the store and part of it needed. Establish the library in the store by starting at 12 and then proceed as described under (1).

3.5 THE LIBRARY

The standard library contains the following procedures:

arctan
cos
sin
sqrt
} stored under library name QATRIG

instring
outstring
} stored under library name QASTRI

lowbound
range

A reference to sqrt causes loading of the sqrt procedure, but a reference to cos causes the procedures cos and sin to be loaded. The mechanism which selects a portion of the library for copying to the end of the Algol program works as follows:-

A complete procedure is read into store from the left hand global name which starts it until the checksum which closes it; it is then copied or rejected. If an end-of-library mark is read then the library scan finishes.

3. 5. 1 Structure of the Library

Basically the library tape consists of a series of independently assembled SIR blocks and is in relocatable binary format. Each block begins with a left-hand global label and ends with a checksum.

Following the last block on the library tape is a single character for halting the library scan. Between each block is a length of blank tape.

3. 5. 2 Adding procedures to the library.

Providing that there is no name clash, the procedure written in SIR code should be assembled with option +0 and the resulting relocatable binary tape inserted by a copying process (see 3. 4. 11. 1). These procedures must still be declared in the Algol text as described in 3. 4. 1.

The standard version of the library is "built-in" to the Interpreter (Tape 2). The names in the standard library, with suitable declarations, are "built-in" to the Translator (Tape 1). New versions of the Library may be "built-in" as described in Chapter 5. 5.

3. 5. 3 Lowbound

This library procedure is used to determine the lower subscript bound of a real array. Together with the procedure range it enables matrix procedures to be written without requiring the bounds of the array parameters to be handed over as extra parameters.

The implied declaration of lowbound is:-

```
integer procedure lowbound (A, n);
      value n; array A; integer n;
```

Comment 'A' may have any number of dimensions. The procedure takes the value of the lowerbound of the nth bound-pair;

3. 5. 4 Range

The implied declaration of range is:

```
integer procedure range (A, n);
      value n; array A; integer n;
```

Comment 'A' may have any number of dimensions. The procedure takes the value of the range of the nth bound-pair, i. e. (Upper bound-lower bound +1) e. g. with array A [1:8, - 1:3] range (A, 2) will take value +5;

3. 6 Control Procedures

These are parameterless procedures which help to control the running of the program.

3. 6. 1 STOP

This procedure causes an immediate ending of the program, as when the final end is reached. FINISH is displayed and the standard trailer output.

3. 6. 2 WAIT

This procedure causes the program to enter a Wait stop. The program may be continued by entry at 9. It does not affect the calculation in any other way. Operation is frequently simplified if appropriate instructions are displayed immediately before wait is called, e. g.

print punch(3), "L' LOAD NEXT DATA TAPE'; wait;

3. 7 Procedures for Character Input

There are three procedures on the 900 ALGOL Tape 3 Library tape which are concerned with input of individual characters. These procedures are not "built-in" to the standard ALGOL Tape 2, and therefore a Library scan is required to include them. The procedures are Advance, Buffer and Decode, described below by their implicit declarations. (They do not have to be declared as such in the Algol Program.)

1) ADVANCE

Code procedure ADVANCE (I); Value I; Integer I;
ALGOL;

Comment This procedure reads one character from device I into a buffer. In the standard system, devices 1 (paper tape input) and 3 (teleprinter input) are allowed. This buffer is used by the Buffer and Decode procedures; also by read statements. ;

2) DECODE

Code integer procedure DECODE (I); Value I;
integer I; ALGOL;

Comment This procedure examines the character buffer input device I, and delivers as an integer result, the 900 internal character code value in the range 0 to 63. See the Facts Card or 900 SIR manual (Volume 2. 1. 1) for details of internal code values. DECODE does not itself cause input of a character, it must be preceded by an ADVANCE or a read. If a read or an instring from the same input device follows, the first character processed will

be the character examined by DECODE (or BUFFER). If DECODE follows a read or instring,the character examined will be the character which follows the last digit or decimal point; or the character following close quote for an instring:

3) BUFFER

```
Code boolean procedure BUFFER (I, S); Value I;
        integer I ; string S; ALGOL;
```

Comment Procedure BUFFER takes the value true if the character represented in string S is the character in the buffer for device I. Otherwise,it has value false. The string corresponding to S must consist of three characters only; the open quote, the character to be compared, and the close quote sign. It must not contain inner string characters, so that BUFFER (1,'' L'') is not allowed, it must be written on two separate lines, as:

```
BUFFER (1,'
')
```

The rules mentioned under DECODE for use before or after read or instring apply to BUFFER ;

Example of program using ADVANCE, BUFFER and DECODE.

```
PROG;
 begin
   integer array A [ 1:100 ] ; array B[ 1:100];
   integer I, J, K, IDEV, CHAR; real X;
   switch SS:=L1, L2, L3;
   read READER(3), IDEV;
   READER (IDEV); K:=1;
L1: ADVANCE (IDEV);
   if BUFFER (IDEV, '*')
     then STOP;
   comment * marks the end of the data tape;
   CHAR:=DECODE(IDEV);
   if CHAR=13 or CHAR=14 or CHAR=11
     or (CHAR > 15  and CHAR<26)
   then goto L2; comment if -. + or digit;
```

```
        if CHAR=7 then goto L3; comment string quote;
        goto L1;
L2:     read I, B[I];
        goto L1;
L3:     INSTRING (A, K);
        goto L1;
```

Chapter 4: ERROR INDICATIONS

4.1 ERRORS DURING TRANSLATION OF PROGRAM

4.1.1 Run in Translation Mode (start at 8 or 12).

If an error is detected during translation, a message is punched separated from the binary output by twenty blanks. No further binary output occurs, but scanning of the input text continues to check the remainder of the program for further errors.

The form of an error message is,

```
ERROR NO 28
LINE NO    6
  2:=6;      ANS:= 4 * A2;
     ↑
```

The error number given in the message refers to the table in 4.1.5, and the line number is the count of lines on which printable material has occurred from the beginning of the Algol text. The first line of the title counts as line number 1. The line at which the error was discovered is displayed. The error will usually be found in this or one of the immediately preceding lines. Certain declaration errors, such as omitted or mis-spelled identifiers, will however not be detected until the entity in question is referred to, perhaps many pages later.

If an error occurs in a declaration then it is likely that some spurious error messages will be produced later on. Any error may give rise to further spurious error messages.

An illegal or odd parity is replaced by the character ←; error 98 is given but no arrow is printed underneath in this special case. Thus several spurious characters can be displayed.

4.1.2 Run in report Mode (start at 10).

The first output consists of a copy of the first six letter of the title of the program and this is followed by

Error message, if any, as described above
Warning messages
Report messages

4.1.2.1 Warning Messages

There are two sorts of warning message which are

```
*WARNING          Produced by end followed
 LINE NO 25       by a comment containing
  "END" X:=1;     a delimiter.
```

and

```
*WARNING          Produced by an identifier
 SUM              declared but not used.
```

It is to be noted that the second type of warning message may be produced if a formal parameter is blocked by a later declaration

```
e.g.   real procedure P (a); real a;
           begin integer a, b;
                 b:= a;
```

In this case the warning is given at the end of the procedure body. This warning is not given for a switch identifier.

4.1.2.2 Report Messages.

The purpose of these is to help the user to locate a fault at run time by giving him information about labels and procedures.

When a label, a procedure declaration or an "END" is encountered a message is punched, with the respective forms:

```
L LOOP ADR  18 (position of label LOOP)
P CAT  ADR  35 (position of procedure CAT (A, B, . . . .))
E      ADR  50 (position of any "END")
```

The address (ADR) is the object program address (relative to its start).

At run time when an error occurs the relative address is given, as is the address to which a return is to be made after calling a procedure. If therefore a parameter mis-match is found at the entry to a procedure, the fault can be traced to the point of call.

If any error message has occurred then the word FAIL is punched followed by a halt code, twenty blanks and a visible X. This is followed by the standard trailer of 50 blanks, 6 erase signs, 100 blanks. Tape is read and ignored up to the next halt code.

4.1.3 Error During Library Scan.

If a library program exceeds the buffer space during the library scan then the FAIL sequence is punched as above. This can only happen if a library program written by a user is longer than about 600 words, or more than about 50 declarations occur in the outermost block of the program.

4.1.4 Undetected Errors.

4.1.4.1 Arithmetic and boolean expressions are indistinguishable between if and then. For example

if a + b then is not distinguished from
if a + b ≠ 0 then

4.1.4.2 Actual parameters are checked against formal ones at translation time except for parameters to formal procedures. The check on these is deferred until run time.

4.1.4.3 A jump into a block is an error detected only at run time.

4.1.5 Error Table at Translation Time.

1	read misplaced
2	print misplaced
3	Constant or expression in read list.
4	Wrong delimiter in switch declaration.
5	Illegal actual parameter
6	Too many parameters to a procedure.
7	Illegal number.
8	Integer number too big.
9	Two statements in the same block are prefixed by the same label.
10	Identifier or constant not as expected.
11	Letter, digit, "." or "$_{10}$" misused.
12	true or false follows an identifier or constant.
13	comment does not follow ";" or begin
14	<reserved>
15	Unrecognised basic symbol.
16	No assignment to the procedure identifier occurred within the body of a type procedure.

17	An identifier in the value or specification part of a procedure is not a formal parameter.
18	Use of undeclared identifier, or label in an inner block.
19	Illegal symbol.
20	Non procedure identifier used as a statement.
21	":=" omitted from for clause.
22	Illegal use of label name.
23	Inadmissible array declaration
24	<switch name> not an actual parameter nor preceded by goto
25	Non type procedure as function designator.
26	switch misplaced.
27	Declaration without identifier.
28	":=" preceded by a constant or used inside an expression.
29	":" in type or switch declaration or misused.
30	Adjacent delimiters inadmissible.
31	Constant before ":=" or "[", or constant or name of a string in a read list.
32	Item other than a non-type procedure used as a statement.
33	Identifier or constant follows a closing round or square bracket.
34	Relation on each side of a simple arithmetic expression.
35	Illegal statement, delimiter misused.
36	Declaration starts incorrectly.
37	Error between for and ":=".
38	Missing array or switch name or "[" misplaced.
39	";" misused in array declaration.
40	end misused.
41	Local identifier used in array bound.
42	goto follows an identifier or a constant.
43	Wrong for clause preceding do
44	for misused.
45	Misused Boolean constant.
46	Assignment to procedure identifier outside procedure body.
47	real integer or boolean misplaced.
48	Identifier declared twice in same block-head.
49	Blank parameter.
50	No begin at start of program.
51	Wrong number of subscripts or parameters.
52	":=" appears in an actual parameter list.
53	Statement ends incorrectly.
54	Declaration follows statement.

55	':', go to or for used in expression.
56	Illegal parameter comment or) precedes identifier.
57	Wrong use of delimiter.
58	Relational or logical operator used as an arithmetic operator.
59	Illegal use of logical operator.
60	Omission or error precedes begin or begin follows ":="
61	"(" misplaced or missing procedure name.
62	Function designator as designational expression.
63	Misplaced declarator.
64	Subscripted variable as statement.
65	Illegal specifier.
66	Misused comma or colon in an expression.
67	if misused.
68	if used in type declaration.
69	Corresponding if has been omitted or conditional expression without an else
70	Corresponding then missing.
71	Illegal character in inner string. (May be caused by missing ` (close quote) in a previous string.)
72	array misplaced.
73	Left square bracket not preceded by an identifier.
74	Unmatched closing square brackets.
75	Upper bound missing in array declaration.
76	Illegal type declaration.
77	Illegal array list.
78	Corresponding for missing.
79	A jump is made to a label declared, but not placed in the block that ends here.
80	step until or while misused in for list element.
81	Misused ")" other than in expression.
82	")" misplaced or unmatched.
83	Program too complex, i.e. some statement is too complicated.
84	Wrong delimiter after procedure statement.
85	Program too large, i.e. contains too many names, labels, constants or switches.
86	Error before procedure.
87	Repeated formal parameter.
88	Wrong formal parameter delimiter.
89	<reserved>
90	Wrong delimiter in value or specification part.
91	Input buffer overflow, i.e. more than 120 characters in a line.
92	Formal parameter has not appeared in the specification part.

93	Declaration terminated by end or containing begin
94	A formal parameter which is a switch, string or procedure is called by value.
95	Switch designator has more than one subscript
96	Wrong for clause.
97	then misused.
98	Illegal character or parity error. The character is replaced by ← in the displayed line, but ↑ is not printed beneath it.
99	Current use of identifier inconsistent with previous uses.
100	Conditional expression needs parentheses.
101	Wrong delimiter after procedure identifier in procedure declaration.
102	No ";" between formal parameter part and value or specification part.
103	Commas or colons wrong in array bounds.
104	div used with a real argument
105	Illegal parameter delimiter after a string.
106	Integer labels not allowed.
107	Recursive function calls not allowed.
108	An actual parameter which is a procedure has one of its parameters called by value.
109	Constant should not be used in procedure heading.
110	Wrong specification part.
111	Different number of parameters from previous use of formal procedure or wrong number of subscripts.
112	Mixed types in multiple assignment.

Note: The following errors will cause an Algol program tape to shoot through and unload the reader, instead of stopping at the end;

(1) No halt code at the end of a tape which is not the last tape of the program

(2) Insufficient end's to match all the begin's in the program.

(3) Missing ` at the end of a string, causing the program statements that follow to be treated as part of the string.

The following errors will cause the end of an Algol program to be found prematurely:

(1) Missing begin.

(2) end or the comment following end not followed by end, else or a semicolon causing a begin to be treated as comment. These errors lead to a breakdown of the block structure of Algol and will usually cause many error messages to be displayed.

4.2 ERRORS DURING LOADING OF THE PROGRAM.

When the program is loaded certain errors are detected by the loader and cause a message to be printed. These are:

FA, FD, FF } misread or mispunched tape.

FC implies that the library names are not an exclusive set and that two library programs with the same name have been copied onto the end of the Algol object program tape. This error is also given if the user entitles his program "COS" or some other library name and calls for the same name in his Algol text.

FU The name printed beside FU is that of a missing library function which should have been supplied during the library scan part of translation. This would occur if the name of an additional library procedure had been mis-spelled in an Algol program.
The missing procedure may be read in at run-time.

FE indicates Store Full. The program is too large to be loaded in its present form. If the program was loaded by entry at 8 it should be re-translated in the library mode (entry at 12) and the program loaded by entry at 13. If FE is given when the program is loaded by entry at 13 the Algol text should be studied to see if the program size can be reduced, or split into two parts.

The procedures in the standard Algol library are

arctan
cos
sin
sqrt
instring
outstring
lowbound
range

Many establishments will have special library tapes containing additional procedures whose bodies are written in code. Programs using such tapes must also avoid clashes of name with any of these additional procedures.

In addition to reporting errors the loader lists the addresses of library labels and indicates the extent of store used by the object program. For example, a program entitled FRED which requires ARCTAN and COS from the library causes a print out like this:

FRED	4000	
QATRIG	4323	} Shared library program.
ARCTAN	4418	
COS	4423	
SIN	4428	
QACODL	4616	Start of object data load (constants etc)
QAVNDA	4650	Start of data area for scalars
FIRST NEXT		
4000	4692	

In this example the interpreter occupies locations 8-3999, the program occupies locations 4000-4691 and the dynamic workspace available for evaluating expressions and entering procedures and for arrays, extends from location 4692 to location 8179.

4.3 ERRORS DURING RUNNING OF THE PROGRAM.

When an error is detected at execution time, 20 rows of blank tape are punched for each error and a report of the form.

ERROR NO	ADR	RET
1	36	20

is displayed.

A halt then occurs from which a restart from the point reached is sometimes possible (see below) depending on which error has occurred. In all cases a restart from the beginning of execution is possible.

Continuing the example of sub-paragraph 4.1.2.2, the block number and address indicate that the error occurred at the entry to procedure CAT and the error number indicates a mismatch between actual and formal parameter types. The call which was responsible for this error is at, or just after, the label LOOP: since the return is to address 20 and LOOP is at address 18. A negative number under ADR shows that the error was detected in a "built-in" library function.

The error numbers and their meanings are given in the table below.

4.3.1 Undetected Errors.

In the special case of a formal procedure whose parameters are being compared with those of an actual procedure, boolean and integer parameters are indistinguishable in all circumstances, thus

boolean, boolean array, boolean procedure } are not distinguished from { integer, integer array, integer procedure

Corresponding errors in parameters to ordinary procedures are, of course, detected at translation time.

4.3.2 Error Table at Run Time.

Error Number	Meaning	Value substituted or Effect on continuation at 9
1	Parameter mismatch	continuation not possible
2	Space overflow, e.g. too much claim on store for an array	continuation not possible
3	Integer overflow	continuation not possible
4	Jump error, i.e. switch subscript outside its permitted range	continuation not possible
5	Subscript error, i.e. address outside the store area allotted to the array referred to	continuation not possible
6	Illegal symbol inside inner string quotes (only discovered when the string is output)	newline is substituted
7	Attempt to output an unstandardised floating point number (Probably the result of an error in a code procedure)	*** is substituted
8	Illegal character or ' found when attempting to read a number	The read routine is re-entered
9	Real overflow	P is substituted
10	Invalid argument for sin(E) or cos(E), i.e. exponent greater than or equal to 18	0.0 is substituted
11	Negative argument for sqrt(E)	sqrt(abs(E)) is substituted
12	Argument >40 for exp(E)	P is substituted
13	Argument $\leq$ 0 for log(E)	0.0 is substituted
14	Illegal character on data tape	space is substituted

Error Number	Meaning	Value substituted or Effect on continuation at 9
15	Parity error on data tape	space is substituted
16	(see Appendix 2 paragraph 3.6)	
17	Numeric character found before when attempting to read a string	The instring operation is ignored
18	Illegal form of number input (e.g. 2.21.3)	Number read so far is ignored, and the read routine re-entered
19	A B with A and B real and A<0	0.0 is substituted
20	Program corrupted, possibly due to error in machine code procedure	Continuation not possible
21	An attempt has been made to assign a value to a formal parameter which is a constant	Continuation not possible
22	Range of array subscript bounds is negative	Continuation not possible
23	Instring or outstring error, i.e. Array not a one-dimensional integer array, subscript value is outside range or instring overfills the array.	Continuation not possible
24	Attempt to jump to a label in an inner block or into a for loop, or label incorrectly declared in an outer block.	Continuation not possible
25	The Translator used for this program is incompatible with this version of the Interpreter.	Continuation not possible.

Notes:

(1) $P = (1-2^{-37}) \times 2^{63}$

except for overflow of a negative real number where

$P = -1.0 \times 2^{63}$

(2) When continuation is not possible entry at 9 results in a dynamic stop.

Chapter 5: OPERATION OF THE ALGOL SYSTEM.

5.1 GENERAL

903 Algol is a two-pass system. In the first pass, one or more programs are translated to paper tape or checked for syntactic errors. In the second pass, the translated programs are loaded and run one at a time.

5.2 TRANSLATION

The binary translator tape (Tape 1) is read in by Initial Instructions and clears the store before being loaded. There are five starting addresses:

8	Normal	
9	Continue	(after halt code etc).
10	Report mode	(gives warning, report and error messages only).
11	Checking mode	(as Normal but CHECK functions included).
12	Library mode	(as Normal but a library scan takes place after translation).
13	Library checking mode	(as 12 but CHECK functions included).
14	Normal mode with reports	(use only if on-line teleprinter fitted; punches intermediate code and prints reports).

After an error has occurred the mode changes to Report mode.

Algol text, which may be offered on several tapes, is translated into a relocatable binary output until either

1. The final end is read.
2. The Algol text shoots through the reader after the final end.

Case 1

In this case a wait occurs. If the standard trailer is 50 blanks, 6 erases and 100 blanks has not been output then the library tape of Algol functions (Tape 3) should be offered and a start made at 9. This causes appropriate sections to be copied in relocatable binary form on to the end of the User's tape and this is terminated by the standard trailer. This will only be required after use of Library or Library checking mode.

For all other starting addresses the output is complete, but may contain a visible X just before the standard trailer. This indicates a failure somewhere and the last piece of tape should be printed.

Case 2

If the Algol text shoots through the reader the program is corrupt. Remove it and proceed to translate the next program.

It is unlikely that an erroneous Algol program will corrupt the translator.

When translation is completed successfully, and the final trailer is output, the input tape is read and copied up to and including the next halt code. This allows small quantities of data to be copied from the end of the source code tape, convenient for input at run time.

5.3 LOADING AND RUNNING

The binary interpreter tape (Tape 2) is read in by Initial Instructions and clears the store before being loaded. There are six starting addresses:

8	Load an object program tape.
9	Continue after a wait.
10	Execute a loaded program.
11	Read in a related relocatable tape.
12	Establish the library in the store.
13	Read in an object program over the library.

Start at 8

This causes the object program tape to be loaded after the end of the library in the store; the entire library is in the store and accessible to the User. If there are library functions on the tape, FC will be output, and the program must be re-loaded at 13.

It may happen that not enough store is available in which case the program should be retranslated in Library mode and loaded by a start at 13 (see below)

Start at 9

This continues after any wait or after a continuable error.

Start at 10

The name of the program is punched and then execution takes place as directed by the User's program.

At the end of the program or when the procedure STOP is executed, the message FINISH is punched and followed by 100 blanks.

Start at 11

This is for supplying a machine coded procedure at run time instead of at translation time. It must have been preceded by a start at 8 or 13.

Start at 12

This establishes the library in the store which is necessary if ever a start has been made at 13. Offer Tape 3 and start at 12. It may be used to load modified versions of the library.

Start at 13

This loads a complete program over the library. This start must be employed if the Algol program had been translated in Library mode and had extracted a program from the library.

During loading a storage map is produced on the punch and any errors which are detected cause messages to appear among the storage map. If a previous program has overwritten the loader then, on starting at 8 or 12 or 13, no input occurs, instead the message

RELOAD TAPE 2 is displayed.

5.4 DUMP facility

The dump facility allows a complete image of the store to be dumped on paper tape. This facility is only available in 903 Algol on the basic paper tape system for 8192 word (8K) stores.

It may be used for three purposes:

(1) To generate copies of the standard Tape 2, or versions of Tape 2 modified by the user; e.g. with a different "built-in" library.

(2) To dump an Algol program which has been loaded into the Interpreter, so that in future the program may be run by loading a single sum-checked binary tape.

(3) A long running program may be stopped, and a dump of the current state of the store taken. The dump can later be re-loaded and the program continued from the point at which it was stopped.

To operate the dump, if an Algol program is actually running set the level 1 switch (on the lower centre of the operator's control panel) to Manual and press the level 1 interrupt button. Then or otherwise enter at 14 to dump the store. A complete paper tape version of the first 8180 words of store will be output. This should be checked at some point by input to the computer under initial instructions (entry at 8181), which is the normal method of loading this tape.

If the program was stopped by interrupt on level 1, it may be re-started by entering at location 9 after the dump is complete. Similarly, the dumped program may be re-started at location 9 after the paper tape is input. Alternatively the program may be started from the beginning by entry at 10, in the normal way.

The dump facility will not work if store above location 8000 has been used for array data. If this store is currently in use NO PROGRAM will be displayed on entry at 14.

The dump facility could be called from a SIR code procedure by storing a return address in 20 and transferring control to location 14 as shown in the following example:

```
        [DUMP]              (declare as:)
DUMP    /14   0             (code procedure DUMP; algol;)
              +0
          4   CONT
          5   20
          8   14
CONT      0   ;+1           (continuation address)
          0   DUMP+1        (On re-entry at 9, exit from DUMP)
         /8   1
          %
```

The dump facility may be used as a means of producing copies of the Interpreter for the basic 8K paper tape system, by simply loading the Tape 2 and entering at 14. These copies should be checked by input (at 8181) in the normal way. Similarly, the Translator (Tape 1) for the basic system may be copied by entering at location 8000. The copies produced in this way will not have the legible titles on the leading end, otherwise they are accurate copies, which may be used to produce further copies.

5.5 Altering the built in library

Chapter 3.5.2 describes how procedures written in SIR code may be added to the library tape. This library tape may be "built-in" to the Interpreter (Tape 2) by loading at 12 and then dumping Tape 2 as described in 5.4.

New names may be added to the built-in name list in the Translator (Tape 1) as follows:

(1) Prepare a tape equivalent to the start of a 903 Algol program, with the required names declared as a series of code procedures, followed by a halt code. "END" must not appear on this tape. See the example below.

(2) Input this tape to the Translator (Tape 1) by entering 10 (Report Mode).

(3) Enter at location 16. A complete version of the Translator will be output on paper tape, with the new names added as declarations to the "built-in" list.

Example:

If two procedures RANDOM and MAX had been added to the library tape, the following code might be used to enter these into the "built-in" Translator list, as in (1) above.

```
TITLE; "BEGIN"
"CODE" "REAL" "PROCEDURE" RANDOM (I);
       "INTEGER" I; "ALGOL";
"CODE" "INTEGER" "PROCEDURE" MAX (A, B);
    "INTEGER" "ARRAY" A; "REAL" B; "ALGOL";
```

(H) (Halt code)

Appendix 1: COMMON ERRORS MADE IN PROGRAM WRITING:

1.1 PROGRAM CHECKING

A program, whether in Algol or any other programming language can be a fairly intricate composition, and it is recommended that a systematic check be made for errors and slips before first running it. The computer does not ignore slips; an apparently trifling error in punctuation may well make a program unacceptable or, what is worse, lead to incorrect results. The following list of checks will indicate the type of mistakes most commonly made.

1.1.1 General Program Checks.

(1) Check that the multiplication symbol * has not been omitted - a common and simple mistake to make and to overlook.

(2) Make sure that no division by zero, no determination of the logarithm of zero or of a negative quantity, no determination of the square root of a negative quantity, and no evaluation of a↑b (a < 0, b non-integral) or any other undefined operation can occur.

(3) Check that all numbers are written in the accepted forms

(4) Check that two arithmetic operators do not appear next to one another.

(5) Check that when testing the magnitudes of quantities the absolute values of these quantities are used.

(6) Check that cycles are nested correctly and that each cycle terminates.

1.1.2 Special Checks for Algol

(7) Check throughout the program that all basic words used (begin, end, for, etc) are underlined.

(8) Check throughout for the correct spelling of basic words.

(9) Check for correct punctuation; in particular, check for the correct placing of semicolons.

Remember that the first symbol following any statement must be either

; or else or end

and that end must be followed by a semicolon, else or another end (intervening words are treated as comment). Failure to put a semicolon after end will result in the next statement being treated as comment and not translated and consequently not executed. Check also that the word comment occurs only after a semicolon or begin, and that the comment material is terminated by a semicolon.

(10) Check that for every if there is a corresponding then. Check also that an if never follows a then. Remember that a conditional expression must always have an else part, although a conditional statement need not. Thus the statement

if A < 0 then A:= - A

is equivalent to

A:= if A < 0 then - A else A;

and it is incorrect to write

A:= if A < 0 then - A;

(11) Check that all variables on the left-hand side of an assignment statement are of the same declared type (e.g. a:= b:= c*d is incorrect if a is real and b is integer).

(12) Check the correct use of brackets; in particular, check that there are corresponding opening and closing brackets in arithmetic expressions, that the argument for a standard function is enclosed in brackets (e.g. sin(x) not sin x) and that square brackets are used in arrays and around suffixes. Remember the natural order of precedence of the operators in arithmetic, or Boolean expressions, and check that brackets have been used to change the order of execution of operations where required.

(13) Check that each begin has a corresponding end.

(14) Check that each compound statement (i.e. a sequence of statements to be executed together as one unit) is bracketed between begin and end. This check is particularly necessary in the case of a compound statement following a for clause or a compound statement constituting a branch of a conditional statement.

(15) Check that all variables are declared and are not used outside the block in whose head they are declared, and that variables are not introduced into expressions before values have been assigned to them. Check also that identifiers differ in their first six symbols. Check that all labels are declared in a switch declaration at the head of the innermost block in which the statement to which they are attached occurs. Check that declarations occur only at the head of a block.

(16) Check that the operator ÷ (or its equivalent form div) is used only when both operands are of type integer. In particular, note that the result of exponentiation i ↑ j is real even when i and j are of type integer, so that (i ↑ j ÷ 2) is incorrect.

(17) Check the correct use of labels in goto statements. No goto statement may lead from outside into a block or from outside a for statement to a statement within the for statement.

(18) For every conditional statement or expression establish two tests, one which makes the arithmetic comparison true, and one which makes it false. By following the action of the program for both these cases check that the program always behaves correctly.

(19) When using a for clause, such as for V:= L do, remember that when the list of values of V is exhausted, the variable V cannot be used again in the subsequent program until it has been assigned a value. V retains its current value only when a goto statement brings about a jump out of the for statement before the list of values of V is exhausted.

(20) Numbers of type real cannot in general be held absolutely accurately in the computer. The error is only of the order of one part in 10^{-8}, but it must be borne in mind when the equality of two real numbers is tested for. Thus the relation A = B may have the value false at a certain point in a calculation even though A and B are theoretically equal. To avoid errors of this sort, write the relation instead as abs(A - B) < epsilon, where epsilon is a suitable small positive constant. This inaccuracy in the computer representation must also be taken into account in for statements. Thus

for A:= 1 step 1/3 until 2 do C;

could result in the statement S being executed for A = 1, $1\frac{1}{3}$, $1\frac{2}{3}$, but not for A = 2. On the other hand,

for A:= 1 step 1/3 until 2 + epsilon do S;

is safe.

The list given here is not exhaustive.

It is good practice to work through the program with a set of test data, to make sure that it behaves correctly at every stage. This is often the best way of locating errors.

Appendix 2: NOTES FOR USER'S OF ALGOL ON 920 COMPUTERS.

1. General.

Two forms of Algol are available on 920 computers:

1.1 903 Algol.
1.2 920 Algol.

A special version of 903 Algol which operates in 503/920 telecode. This special version is known as 920 Algol.

1.3 Mode Switch.

The tape-reader mode switch is referred to below. On certain 920 computers a mode switch is not fitted, or if fitted is marked in a different manner from that described. The modes of operation are therefore defined here (s indicates the sprocket hole).

Mode 1 (Also known as 503 mode)

Track on tape	1 2 3 s 4 5 6 7 8	Bit 5 not used
	↓ ↓ ↓ ↓ ↓ ↓ ↓	
Accumulator	1 2 3 4 5 6 7 8	Bit 8 not affected

Mode 3 (Also known as 4100 mode)

Track on tape	1 2 3 s 4 5 6 7 8
Accumulator	1 2 3 4 5 6 7 8

2. Use of 903 Algol on 920 Computers.

2.1 If a mode switch is fitted to the paper tape station it must be set to position 3.

2.2 Certain 920 tape readers are of a type which does not stop on a character, if a machine with such a tape reader is used the following precautions must be taken.

2.2.1 On program (source tapes) every new line must be followed by three blanks (instead of the one normally recommended).

2.2.2 On data tapes the terminator of every data item must be followed by three blanks.

3. Use of 920 Algol.

3.1 If a mode switch is fitted to the paper tape station it must be set to position 1.

3.2 Certain 920 tape readers are of a type which does not stop on a character, if a machine with such a tape reader is used then;

3.2.1 On both program and data tapes every new line character must be followed by three blanks.

3.2.2 There may not be more than 120 characters on a line of data.

3.3 Character set.

The following special punching conventions apply.

Algol Symbol	Hardware representation 903 Algol	920 Algol	punching method for 920 Algol
' ' begin etc	' ' "BEGIN"etc	⊀ ⊁ ~BEGIN~ etc	vertical bar followed by < or >
additional characters			
	½	¼	vertical bar followed by 2
	$	$	vertical bar followed by S
	←	?	

All other characters are punched as described in chapter 1. No other compound characters are recognised. It should be particularly noted that the non-escaping underline character must not be used and that basic words must be punched in the style described above.

3.4 Tapes

The tapes are marked

920 Algol Tape 1 (Translator)
920 Algol Tape 2 (Interpreter)
920 Algol Tape 3 (Library)

3.5 Error messages at translation time.

Unrecognised characters are replaced by ?
Wrong parity characters are not detected.

3.6 Error messages at run time.

If a line of data contains more than 120 characters error No. 16 is displayed. On continuation the entire line of text is ignored.

3.7 Facilities available in 920 Algol

All the facilities described up to the last paragraph of Chapter 5.3 are available. Copying of data by the Translator, the Dump Facility, and procedures for Optional peripherals, are not available.

Appendix 3: USE OF OPTIONAL PERIPHERALS

1. USE OF THE 903 DIGITAL PLOTTER

A set of procedures are provided which enable a 903 Algol program to use the plotter. Procedures are available for drawing, plotting points, and drawing characters. The procedures are included in the 903 Algol Library tape, and have "built-in" declarations in the Translator, as for SIN, INSTRING, etc. However, unlike the standard library routines, they are not "built-in" to the Interpreter (Tape 2). Programs using the Plotter, therefore, must always be translated in Library mode (entry at 12, see Chapter 5.2) and loaded by entry to the Interpreter at 13.

The facilities provided are a subset of those in Elliott 4100 Algol.

1.1 Drawing and Plotting

Four procedures are available for use in drawing graphs, pictures and diagrams. SETORIGIN must always be used before any other output to the plotter.

1.1.1 SETORIGIN (E, PAGE)

This procedure is used to set the origin and position the pen. It is a non-type procedure, with two integer parameters, called by value.

The origin is set E plotter steps from the left margin of the plotter, and this point is taken as having zero coordinates (O, O). E should be a positive integer, greater than or equal to 10. The pen is left in the raised position.

PAGE determines the orientation of the axes. If PAGE = 0 the X - axis runs across the paper, from West to East, and the y-axis runs from South to North along the paper. If PAGE = 1 the X-axis runs along the paper, and the y-axis across the paper, i.e. the axes are rotated through 90 degrees in an anti-clockwise direction.

If PAGE is not equal to 1 it is taken to be zero. "North" in this context is the direction from the pen towards the roller from which paper normally unwinds. One plotter step is the distance covered by the pen during one increment. The actual length varies with the plotter models.

1.1.2 DRAWLINE (E, F)

This procedure draws a line with the pen down, from the current pen position to the point with coordinates (E, F). It is a non-type procedure with integer parameters called by value. E and F are measured in plotter steps. On completion the pen is left in the lowered position.

1.1.3 MOVEPEN (E, F)

MOVEPEN has a function similar to DRAWLINE except that the pen is moved in the raised position to the point (E, F). On completion the pen is left in the raised position.

1.1.4 CENCHARACTER (N)

This procedure is used to plot points on a graph. It draws a character centred on the current position. On completion of the character the pen is left raised in the original position.

The character drawn is a small + . The procedure is non-type, with one integer parameter called by value. The parameter should have value +1. It is inserted for compatibility with 4100 Algol.

1.2 Printing of Numbers and Text

The digital plotter can be used as an extra output device for "PRINT" statements. Before characters can be drawn the procedure WAY (D, L) must be called. After a call of WAY a call of PUNCH (5) will cause the plotter to be taken as the current output device, and characters for output will be drawn on the plotter.

The character set for output is:

letters	A to Z
figures	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
symbols	+ - ₁₀ * =
space.	

All other symbols including newline are output as an inverted V character (Λ).

The procedures SETORIGIN and WAY must be called before the use of PUNCH(5).

WAY (D, L) is a non-type procedure with two integer parameters called by value. D should have the value zero, and is inserted for compatibility with Elliott 4100 Algol. L determines the size of the characters to be drawn. If L is less than one it is taken to be equal to one, otherwise the size of characters is proportional to L. The size of

characters depends on the plotter step size, useful values of L are in the range 4 to 25, giving widths of L*2.5 plotter steps, approximately.

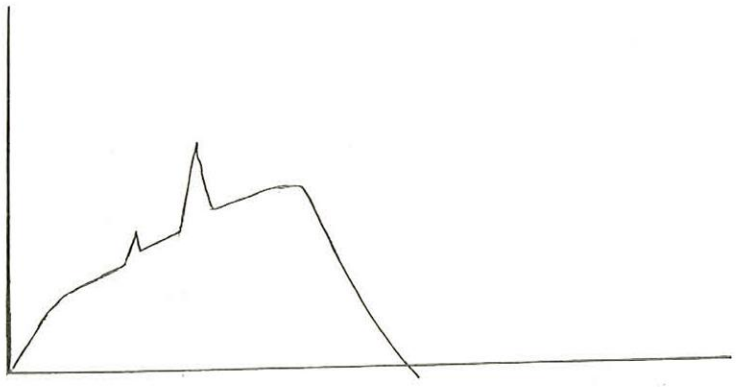
All characters output on the plotter are drawn in a grid of squares whose size depends on the last call of WAY. The initial pen position is at the bottom left-hand corner of the square. The final pen position is at the bottom right-hand corner, ready for drawing the next character. Right in this context is the direction of increasing X, as determined by the last call of SETORIGIN.

1.3 Example

The following program might draw the graph shown:

```
PLOT; "BEGIN" "INTEGER" X, Y;
SETORIGIN (2500, 1); WAY (0, 6);
DRAWLINE (0, 2000); MOVEPEN (4000, 0);
DRAWLINE (0, 0); MOVEPEN (0, -60);
"PRINT" PUNCH(5),´ GRAPH ´S2` X-AXIS`;
MOVEPEN (0, 0);
"FOR"X:=10, X+10 "WHILE" Y "GE" 0 "DO"
"BEGIN" "READ" Y; DRAWLINE (X, 10*Y);
"END";
"END";
```

paper ←



GRAPH X-AXIS

APPENDIX 4: USE OF EXTRA CORE STORE

1. OPERATION of the 903 ALGOL 16K (LG) SYSTEM

1. 1 Purpose

Algol programs may be translated and run "load and go" using this system, which requires 16384 words of core store. The program is assembled in store as it is translated from Algol source code, and is ready for running as soon as the Translation is completed succesfully.

The system is designed for convenience in testing small programs, particularly testing batches of student programs.

1. 2 Store available

The store is allocated as follows:

Algol Interpreter	0 to 4000 (approx.)
Space for program, library routine and data	4000 to 7500
Translator and loader	7500 to 16200

The space available for the largest program is therefore 3500 words. The translator and loader may be overwritten by data (arrays), however, in this case the compiler tape must be re-input before the next program can be run. If batches of programs are to be run, without overwriting any part of the system, the space available for program plus data is 3500 words. Otherwise the space available for program plus data is 12300 words, with program limited to 3500 words, (see 1. 5 below). Larger programs should be run using the 16K (LP) system.

1. 3 Operation

1. 3. 1 Normal operation

(1) Load the tape 903 ALGOL 16K (LG) by Initial Instructions (entry at 8181)

Continuous output on the punch or lighting of the "READ" lamp when the end of tape is reached indicates misread or damaged tape.

(2) Load the Algol program tape and enter at 8. This translates the program and assembles it into one operation.

(3) If the program is continued on further tapes, continue at 9.

(4) To run the program, load any data tape required and enter at 10. The operator should not attempt to run the program if any error messages were output at Translation.

(5) To run further programs, return to step (2). If the message REINPUT TAPE 2 is displayed, return to step (1).

1. 3. 2 Report Mode

If it is likely that the program contains errors, or if an address is required to interpret run-time errors, translate in Report Mode or in Checking Mode (see 1. 3. 3). To translate in Report Mode, at step (2) above enter at 16.

1. 3. 3 Checking Mode

1

To include check functions in the translated program (Chapter 3. 3) translate by entering at 11 in step (2) above. This mode also produces Report Messages.

1. 3. 4 Library Mode.

If the program is too large to fit in the store available with all the "built-in" library procedures, try translation by entry at 12 in step (2) above. When the final end is read, load the Library (Tape 3) and enter at 9.

1. 3. 5 Library Checking Mode

This combines the functions of Library mode and Checking Mode. Enter at 13 to translate in Library Mode, with checks included, and Report Messages displayed. When the final end is reached, load the Library (Tape 3) and enter at 9.

1. 3. 6 Adding SIR code procedures

SIR code procedures must be assembled to paper tape using the SIE assembler, before step (1) above. If they have been added to the library tape (Chapter 3. 4. 11. 1) use Library Mode. (1. 3. 4 above). Otherwise, when step (2) is complete, using any of the Modes described, load the code procedures in translated form and enter at 14. Further code procedures should also be loaded by entry at 14.

1. 3. 7 Re-establish the "built-in" library in store

If further programs are to be run after use of Library Mode the "built-in" library must be restored. Load the Library (Tape 3) and enter at 15.

1. 3. 8 Translation to paper tape for large programs

If a program is too large to run on the Load and Go system; it may be translated to paper tape and run on the Large Program (16K (LP)) system. Load the Algol program tape and enter at 17 to translate to paper tape. This corresponds to entry at 8 to the basic Translator. If check functions are required the basic Translator (Tape 1) must be used.

1. 3. 9 Summary of entry points

8 Normal load-and-go translation
9 Continue after wait
10 Run program
11 Load-and-go translation, with Check functions and Reports
12 Library mode, load-and-go translation
13 Library mode, load-and-go, with Check functions and Reports
14 Load a relocatable binary SIR procedure
15 Re-establish built-in library
16 Report Mode
17 Translate to paper tape (ready for 16K (LP) system).

Volume
Part
Section

1. 4 Error indications

Error indications given are identical to thos e in the basic system, (Chapter 4). If the program is too large for load-and-go operation the loader error FE is output, and translation stops. The 16K (LP) system or Library Mode (see 1. 3. 4 above) must be used to run the program.

If the loader or translator is overwritten by data (arrays) no indication is given, but when attempting to translate the next program the message RELOAD TAPE 2 will be displayed. The compiler must then be reloaded.

Warning messages are only given in the Report Modes (entries at 11, 13, and 16). This avoids slowing down the normal translation. However, if report messages are required all programs may be translated by entry at 11 instead of 8.

1. 5 Use of large arrays

As stated in 1. 2 above, the store from the end of program up to 16300 may be used for arrays. Note that locations 8180 to 8191 will be overwritten in this case. The array data will not be preserved if any entry is made to initial instructions.

2 16k (LP) LOADER/INTERPRETER

2.1 Introduction

2.1.1 Purpose

The 900 ALGOL 16k (LP) Loader/Interpreter is provided to make it possible to run 900 ALGOL programs that are too large for the 8k and 16k (LG) systems.

2.1.2 Configuration

The minimum hardware configuration required for the operation of this system is a 903 or 905 with 16384 words of core store, paper tape reader and punch, and an on-line teleprinter.

2.1.3 Environment

The 900 ALGOL 16k (LP) Loader/Interpreter is designed for use in conjunction with the 900 ALGOL 8k Translator (Tape 1) and the 900 ALGOL Library (Tape 3). (Both tapes at Issue 6 or later.)

2.1.4 Form of Distribution

The 900 ALGOL 16k (LP) Loader/Interpreter is distributed as a single sum-checked binary (SCB) paper tape for input by initial instructions.

2.2 Method of Operation

2.2.1 Translation

The ALGOL program should be translated to a relocatable binary paper tape in accordance with the instructions given in 900 Programming Manual, Volume 2.1.2, Chaper 5.2.

Translation in library mode (including library mode with checks) must not be used.

Any SIR code procedures must be assembled to relocatable binary tape using the standard SIR assembler. (See 900 Programming Manual, Volume 2.1.1, Chapter 3.4.3.)

2.2.2 Loading and Running

Load the 900 ALGOL 16k (LP) Loader/Interpreter tape under initial instructions. This tape contains a built in library (see Chapter 3).

There are six entry addresses for loading as follows:

8 - Load an ALGOL program maintaining existing library and code procedures.
9 - Load an ALGOL program maintaining existing library only.
10 - Load an ALGOL program overwriting existing library and code procedures.
11 - Load a code procedure maintaining existing code procedures.
12 - Load a code prodecure overwriting existing code procedures.
13 - Load the library overwriting existing library and code procedures.

Any entries at 11, 12 or 13 that may be necessary must be made before an entry is made at 8 or 9. This is because the library and SIR code procedures are loaded on level 1 but the ALGOL program is loaded on level 4 and is self triggering. If the ALGOL program is loaded successfully into core a prompt character (←) is output on the control teleprinter. The operator should then type one of the following command letters:

(i) R to run the program

(ii) D to enter the dump facility (see Chapter 2.3)

When a run is complete the word FINISH followed by a further ← is output on the control teleprinter. The operator may then type R or D as before or load further tapes at any of the six entry addresses. An attempt to load further tapes may cause one of the following commands to be output on the control teleprinter.

(i) RELOAD MACHINE CODE

This indicates that part of the SIR code procedure dictionary has been overwritten. Any required SIR code procedures must therefore be loaded at 12 and 11.

(ii) RELOAD LIBRARY

This indicates that the SIR code procedure dictionary and part of the library procedure dictionary have been overwritten. Any required library and SIR code procedures must therefore be loaded at 13, 12 and 11.

(iii) RELOAD SYSTEM

This indicates that both the SIR code and library procedure dictionaries together with part of the loader have been overwritten. Reload 900 ALGOL 16k (LP) Loader/ Interpreter under initial instructions.

2.2.3 The Dump Facility

The dump facility allows a complete image of the core store to be dumped to paper tape for subsequent re-input under initial instructions. There are two methods of entry.

(i) Type 'D' following a prompt character on the control teleprinter.

(ii) After an entry at 13, 12 or 11 but before an entry at 10, 9 or 8 enter at 14. This enables the operator to generate copies of the 16k (LP) Loader/Interpreter with a modified built in library and/or code procedures. Entry at 14 should not be used to dump ALGOL programs.

Dumps produced by method (i) are automatically loaded on level 4 when re-input under initial instructions. Successful loading is indicated by a prompt character on the control teleprinter. The operator should type R to run the program.

Dumps produced by method (ii) are loaded on level 1 and enter a dynamic stop if loaded successfully.

2.2.4 Increasing the Store Available for Arrays

(i) If no library or SIR code procedures are required load the ALGOL program at 10.

(ii) If only SIR code procedures are required load the first at 13 the second at 12 and subsequent ones at 11.

(iii) If library procedures are required the size of the library may be reduced by selectively copying sections of the standard library. (Note: SIN and COS must go together under the name QATRIG, and INSTRING and OUTSTRING under the name QASTRI.) The number of blanks between each procedure and before the final terminating character is not significant.

2.2.5 Error Indications

Error indications are as for the 8k ALGOL System with the following additions:

(i) If on entry at 13, 12 or 11 an attempt is made to load a library or SIR code procedure beyond location 8179↑0 the message 'FZ' is output on the control teleprinter.

(ii) Standard error messages are followed by the prompt character (←) on the control teleprinter. The operator should then type one of the following command letters:

(a) C to continue

(b) D to enter the dump facility (see Chapter 2.3).

N. B. If C is typed following a non-continuable error the prompt character is output again.

2.3 The Library

The 900 ALGOL 16k (LP) Loader/Interpreter contains a built in library consisting of the following standard procedures.

DECODE	SQRT
ADVANCE	LOWBOUND
BUFFER	RANGE
SIN	OUTSPRING
COS	INSTRING
ARCTAN	

2.4 Store Allocation

The Interpreter occupies core locations from 0 to 4400 approximately. The library and SIR code procedures are loaded from location 4400 upwards but cannot extend beyond location 8179. The loader is approximately 750 words long and is located at the top of Module 1. The library procedure, SIR code procedure and ALGOL program dictionaries extend downwards from the beginning of the Loader but, together with the Loader, may be overwritten at run-time. The ALGOL program can extend from end of the SIR code procedures to the beginning of the ALGOL program dictionary but must not exceed 8192 words in length.

APPENDIX 5: USE OF NON-STANDARD PERIPHERAL DEVICES

1. INTRODUCTION

This document describes how additional peripherals (other than standard reader, punch and teleprinter) may be addressed by standard "read" and "print" statements in an Algol program.

2. METHOD

2.1 Introduction

The non-standard device should be assigned a device number, N, in the range 1-10, not already assigned to a standard device. (See Table 1).

The user must write a device routine for the non-standard device together with a Machine code set-up procedure to load the address of the device routine into the appropriate table in the Algol Interpreter. This procedure may, but need not, have one or more parameters to control input/output from/to the device.

The device routine and the set-up procedure will normally be part of the same SIR block and may, if required, be added to the standard Algol Library. (See 900 Manual Volume 2.1.2 Section 3.5.2).

To address a non-standard device an Algol program must firstly make a call of the appropriate set-up procedure. Thereafter the device may be selected by the standard statements, reader (N) and punch (N).

2.2 Set-Up Procedure for an InputDevice

The set-up procedure for input device, number N must;

1) Load +0 into Location (BUFFER+N-1) in the Algol Interpreter if location (INSLOT+N-1) does not already contain the address of the device routine.

2) Load the address of the device routine into location (INSLOT+N-1) in the Algol Interpreter.

2.3 Set-Up Procedure for an Output Device

The set-up procedure for output device, Number N, must:

1) Load the address of the device routine into

location (OUTSLT+N-1) in the Algol Interpreter.

2.4 Device Routine Entry and Exit

Output device routines will be entered with the character to be output, in SIR internal code, in the A-register.

Input device routines should exit with the character input, in SIR internal code, in the A-register.

Exit from input and output device routines must be via the Interpreter link locations ICHLNK and PCHLNK respectively.

TABLE 1

900 Algol - Device Numbers

Number	Device
1	Paper Tape Reader/Punch
2	Reserved
3	Teleprinter
4	Lineprinter
5	Plotter
6	Card Reader/Punch
7	Not allocated
8	Not allocated
9	Not allocated
10	Not allocated

TABLE 2

Example of an InputDevice Set-Up Procedure

```
*0                                          (Card Reader Routine)
[CARDIN]
FP=138
ICHLNK=229
BUFFER=231
INSLOT=241

CARDIN /14 1                                (one parameter)
       +0
       0   FP
       /4  3
        5  BUFFAD                           (Parameter is buffer address)
        4  INSLOT+5
        2  ENTRYA
        0  CARDIN+1
       /7     1
        4  ENTRYA                          (Store device routine address)
        5  INSLOT+5                        (Card reader device number 6)
        4  +0                              (Clear Interpreter Input buffer)
        5  BUFFER+5
       /8     1

ENTRYA 0   ENTRY

ENTRY  4   BUFFAD                           (Device Routine)
                 |
      (Read Card into users array buffer one column at a time and
       convert to SIR internal code equivalent.)
                 |
EXIT    0  ICHLNK                           (Device routine exits with
       /8     1                             SIR internal code character
                                            in A-register.)
```

inserted 27.10.72

TABLE 3

Example of an Output Device Set-Up Procedure

```
*0
[LPRINT]                                    (Lineprinter Routine)

PCHLNK=230
OUTSLT=251

LPRINT /14   0                              (No parameters)
       +0
        4    ENTRYA                         (Store device routine address)
        5    OUTSLT+3                       (Lineprinter device number 4)
        0    LPRINT+1
       /8    1
ENTRYA  0    ENTRY
ENTRY   5    CHAR                           (Device Routine entered with
                                             SIR internal code character
                                             in A-register.)
             :
             :
             :
       (Store character in local
       buffer and output when
       CHAR is linefeed)
             :
             :
             :
EXIT    0    PCHLNK
       /8    1
```

TABLE 4

Fixed Locations in the Algol Interpreter

Issue 5 - 8k, 16k(LG) and 16k(LP)

FP	= 38
ICHLNK	=129
PCHLNK	=130
BUFFER	=131
INSLOT	=141
OUTSLT	=151

Issue 6 - 8k

FP	=138
ICHLNK	=229
PCHLNK	=230
BUFFER	=231
INSLOT	=241
OUTSLT	=251