A MANUAL OF GIER ALGOL 4

developed by

Tove Asmussen, Jørn Jensen, Søren Lauesen, Paul Lindgreen,
Per Mondrup, Peter Naur, and Jørgen Zachariassen

Third edition of A Manual of GIER ALGOL

by

Peter Naur

A/S REGNECENTRALEN . COPENHAGEN

1967

CONTENTS

The Algol 60 Report.

Throughout the present Manual reference is made to the Algol 60 Report or the Revised Algol 60 Report. The differences between these two documents are slight and do not influence the numbering of sections. The full references of these reports are as follows:

J. W. Backus, et. al., Report on the Algorithmic Language Algol 60 (ed. P. Naur), Numerische Mathematik 2 (1960), pp. 106-136; Acta Polytechnica Scandinavica: Math. And Comp. Mach. Ser. no. 5 (1960); Comm. ACM 3 no. 5 (1960), pp. 299-314.

J. W. Backus, et. al., Revised Report on the Algorithmic Language Algol 60 (ed. P. Naur), Regnecentralen, Copenhagen (1962); Comm. ACM 6 no. 1 (1963), pp 1-17; Computer Journal 5 (1963), pp. 349-367; Numerische Mathematik 4 (1963), 420-453.

Other reports relevant to Gier Algol.

(1) P. Naur: The Design of the Gier Algol Compiler. BIT 3 (1963) 124-140 and 145-166; also in Annual Review in Automatic Programming 4 (ed. R. Goodman), Pergamon Press 1964.
(2) P. Naur: Checking of Operand Types in Algol Compilers. BIT 5 (1965) 151-163.
(3) J. Jensen: Generation of Machine Code in Algol Compilers. BIT 5 (1965) 235-245.
(4) P. Naur: The Performance of a System for Automatic Segmentation of Programs Within an Algol Compiler (Gier Algol). Comm. ACM 8 (1965), 671 677.
(5) P. Naur: Program Translation Viewed as a General Data Processing Problem. Comm. ACM 9 (1966), 176-179.

INTRODUCTION.

The present book is the users' manual of the Algol 60 compiler system for the Gier computer known as Gier Algol 4. This system was developed during 1965 - 67 and is a further development and revision of the system described in a Manual of Gier Algol III, distributed in 1964.

Like the previous versions, Gier Algol 4 is based directly on Algol 60, and the Revised Report on the Algorithmic Language Algol 60 must be regarded as the primary definition of the programming language. For this reason the numbering of sections in the present manual continue those of the Algol 60 Report.

The differences between the present new system and the previous version are so numerous and extensive that the manual has had to be rewritten in all of its parts. Very briefly the more important changes are: (1) Integer variables are represented as 40-bit fixed point numbers, with a corresponding gain in range and speed of operation. (2) Patterns of 40 bits may be manipulated freely and at high speed, by special operators. (3) Case expressions and statements, first suggested by C.A.R. Hoare in Algol Bulletin 18, 1964, are admitted. (4) The selection of input for input and output has been made more flexible. (5) A means of ascertaining the available machine configuration is included. (6) The standard output procedures have been revised and a simple, fast procedure added to the set. (7) The input procedures have been overhauled for greater speed and flexibility. (8) It has been made possible to store elements of arrays in the buffer store, thereby increasing the capacity for variables by a factor of about 6. (9) Subscription has been revised, for higher speed. (10) Means for communicating with semi-permanent data areas on a backing disk storage unit are included. (11) Machine language may be written within a program. (12) The check during translation has been extended to include actual parameters, in most cases. (13) The source program may be formed by combining texts from several places and media during translation. (14) Both the translator and the translated program may be stored on any of several media.

In writing the present manual an attempt was made to follow the definitions of IFIP-ICC Vocabulary of Information Processing, first English language edition, 1966. This, in several cases, proved to be a definite help.

The manual was typed by Kirsten Andersen, who also contributed excellent quality punching of the programs of the system itself. Her help is gratefully acknowledged.

6. 8-CHANNEL PUNCH TAPE CODE AND FLEXOWRITER KEYBOARD.

6.1. PRINTING GRAPHIC CHARACTERS.

Lower case	Upper case	Code	Lower case	Upper case	Code
a	A	, oo . o,	w	W	, o ,oo ,
b	B	, oo . o ,	x	X	, oo .ooo,
c	C	, ooo . oo,	y	Y	, ooo. ,
d	D	, oo .o ,	z	Z	, o o. o,
e	E	, ooo .o o,	æ	Æ	, ooo . ,
f	F	, ooo .oo ,	ø	Ø	, o oo. oo,
g	G	, oo .ooo,	0	^	, o . ,
h	H	, oo o. ,	1	∨	, . o,
i	I	, oooo. o,	2	×	, . o ,
j	J	, o o . o,	3	/	, o . oo,
k	K	, o o . o ,	4	=	, .o ,
l	L	, o . oo,	5	;	, o .o o,
m	M	, o o .o ,	6	[	, o .oo ,
n	N	, o .o o,	7	]	, .ooo,
o	O	, o .oo ,	8	(	, o. ,
p	P	, o o .ooo,	9	)	, oo. o,
q	Q	, o oo. ,	,	₁₀	, ooo. oo,
r	R	, o o. o,	.	:	, oo o. oo,
s	S	, oo . o ,	-	+	, o . ,
t	T	, o . oo,	<	>	, oo . o,
u	U	, oo .o ,	_	|	, o.oo ,
v	V	, o .o o,			

The key for _|does not advance the carriage.

6.2. BLANK.

BLANK , o . ,

6.3. CONTROL CHARACTERS.

LOWER CASE , oooo. o , UPPER CASE , oooo.o ,
CAR RET ,o . , TAB , ooo.oo ,
STOP CODE , o. oo, TAPE FEED , oooo.ooo, PUNCH ADRES ,o . ,
PUNCH OFF , o o.ooo, PUNCH ON , o o.o , AUX CODE , o.o ,
PUNCH ADRES and AUX CODE insert their respective codes when depressed simultaneously with any other key.

6.4. FLEXOWRITER KEYBOARD.

START READ | STOP READ | PUNCH ADRES | AUX CODE | STOP CODE | TAPE FEED

TAB | PUNCH OFF | × 2 | / 3 | = 4 | ; 5 | [6 |] 7 | (8 |) 9 | ^ 0 | ∨ 1 | | _ | PUNCH ON

Q W E R T Y U I O P > CAR
q w e r t y u i o p < RET

LOWER CASE: A S D F G H J K L Æ Ø — a s d f g h j k l æ ø

UPPER CASE: Z X C V B N M ₁₀ : + — z x c v b n m , . -

6.5. NUMERICAL REPRESENTATIONS.

In the following table the characters have been arranged according to the numerical equivalent of the hole combination according to the rule of section 10.3. The first column gives the decimal value of the character, the second and third columns give the lower and upper case character, respectively, and the fourth column contains a G in the cases where the character is available only in Gier, but not on the flexowriter.

	LOWER	UPPER	
0	SPACE		
1	1	V	
2	2	×	
3	3	/	
4	4	=	
5	5	;	
6	6	[	
7	7	]	
8	8	(	
9	9	)	
10	(NOT USED)		
11	STOP CODE		
12	END CODE		
13	å	Å	G
14		|	
15	(NOT USED)		
16	0	∧	
17	<	>	
18	s	S	
19	t	T	
20	u	U	
21	v	V	
22	w	W	
23	x	X	
24	y	Y	
25	z	Z	
26	(NOT USED)		
27	,	₪	
28	CLEAR CODE		
29	RED RIBBON		G
30	TAB		
31	PUNCH OFF		

	LOWER	UPPER	
32	-	+	
33	j	J	
34	k	K	
35	l	L	
36	m	M	
37	n	N	
38	o	O	
39	p	P	
40	q	Q	
41	r	R	
42	(NOT USED)		
43	ø	Ø	
44	PUNCH ON		
45	(NOT USED)		
46	(NOT USED)		
47	(NOT USED)		
48	æ	Æ	
49	a	A	
50	b	B	
51	c	C	
52	d	D	
53	e	E	
54	f	F	
55	g	G	
56	h	H	
57	i	I	
58	LOWER CASE		
59	.	:	
60	UPPER CASE		
61	SUM CODE		
62	BLACK RIBBON		G
63	TAPE FEED		
64	CAR RET		

7. THE RELATION BETWEEN GIER ALGOL AND ALGOL 60.

7.1. CHARACTER REPRESENTATION OF PROGRAMS.

The basic symbols of Algol 60 are represented with the aid of the character set given in section 6. Sequences of characters which do not represent Algol symbols according to the rules below will produce an alarm during translation.

There is a choice of input medium and the possibility to let the program text consist of pieces taken from several media. This is further explained in section 13.2.1.

7.1.1. Single character symbols.

7.1.1.1. Letters and digits. Gier Algol adds the letters

æ Æ ø Ø

to the reference alphabet. The appearance of all letters and digits may be seen from section 6.

7.1.1.2. Delimiters. As apparent from section 6 the following simple reference language symbols are directly available in Gier Algol:

\+ - × / < = > ∨ ∧ , . ₁₀ : ; ( ) [ ]

7.1.2. Compound symbols.

Compound symbols must appear exactly as shown in this section, without additional characters such as BLANK or CARRET.

7.1.2.1. Underlined words. Underlined words are produced in Gier Algol by depressing the underline (_) key immediately preceding each letter of the word. The symbols are the following:

true false go to if then else for do step until while comment begin end own Boolean integer real array switch procedure string label value

Boolean and boolean may be used interchangeably. Also go to, goto, and go to.

7.1.2.2. Compound symbols similar to reference language. The following compound symbols, most of which are produced by combining the underline (_) or stroke (|) with other characters, are similar to those of the reference language:

< > ≠ ≡ :=

7.1.2.3. Compound symbols differing from reference language. The following compound symbols show a noticable deviation from the reference language:

Reference language	↑	¬	␣	′	″	÷	⊃
Gier Algol	^	-,	₂	≪	≫	:	=>

7.1.2.4. Extensions of Algol 60. Add the following:

Basic symbol	Reference	Class
abs	7.6	arithmetic operator
entier	7.6	- -
round	7.6	- -
mod	7.6	- -
shift	7.7	pattern operator
case	7.14, 7.15	sequential operator
of	7.14, 7.15	- -
core	12.1	escape symbol
code	12.1	- -
message	7.1.7	separator
¢	9.5	layout bracket
1 2 ... 40	7.7	separator
copy	13.2.1	medium selector
finis	13.2.1	- -

7.1.3. Other characters in input.

7.1.3.1. CAR RET cannot be used between the characters of an identifier or a number.

7.1.3.2. The reaction of the translator to the input characters not already covered is given in the table below. Three contexts are distinguished: (1) Before first begin. (2) Within a string. (3) In any other program context. In addition the possible copying done as message (section 7.1.7) and as prelude and line output (section 13.2.5) is indicated. The reaction denoted char is an error reaction giving the text: character (appendix 3).

Character	Before begin	In string	In program	Message, prelude, line
10 (not used)	char	char	char	ignored
11 STOP CODE	ignored	ignored	ignored	copied
12 END CODE	see section 7.1.5			ignored
13 å Å	letter	letter	char	copied
15 (not used)	char	char	char	ignored
26 (not used)	char	char	char	ignored
28 CLEAR CODE	see section 7.1.4			copied
29 RED RIBBON	char	char	char	ignored
30 TAB	ignored	ignored	ignored	ignored
31 PUNCH OFF	see section 7.1.6			copied
42 (not used)	char	char	char	ignored
44 PUNCH ON	see section 7.1.6			copied
45,46,47(not used)	char	char	char	ignored
61 SUM CODE	see section 7.1.4			copied
62 BLACK RIBBON	char	char	char	ignored
63 TAPE FEED	ignored	ignored	ignored	ignored
65-126	char	char	char	ignored
127	ignored	ignored	ignored	ignored
Parity error	char or stop depending on machine			ignored

7.1.4. Character check sum

During input of the program to the translator, a check sum of all characters is formed. This may be used to check the currect transcription and reading of characters. The check is performed every time the character 61, SUM CODE, is encountered. This character causes the next following character in the input to be compared with a check character, formed as follows:

1) The input sum is put to zero each time CLEAR CODE is met in the input, and at the end of the SUM CODE action.

2) Each character in the input causes the input sum to be increased:

input sum := input sum + 1 + character

3) To form the check character, use the value of the input sum which excludes the SUM CODE itself and compute:

check character := (input sum + input sum : 32) mod 32 + 31

If the check character formed in this way differs from the one found in the input, the error message

sum

is given.

7.1.5. END CODE and pause.

In any context the END CODE character will cause the message

line <line number> pause

to be output on the operator typewriter, whereafter the machine will stop. To continue the translation, the operator must type BLANK on the typewriter.

7.1.6. PUNCH OFF and PUNCH ON.

With respect to the treatment of these characters the translator may operate in two modes, selected when the translation is initiated, see section 13.2.6. In one mode the two characters cause output on the error output medium of the messages

line <line number> off

and

line <line number> on

respectively, and the program text following any PUNCH OFF character, up to the first following PUNCH ON character, is ignored by the translator. In the include-off-on mode PUNCH OFF and PUNCH ON are ignored.

7.1.7. Messages to the operator.

To enable the programmer to give guidance to the operator during program translation, the following facility for output of messages is included. Wherever the delimiter comment is permitted in the program it may be replaced by message. During the first phase of the translation this will cause the following characters, up to the first semicolon, to be output on the alarm message medium (cf. section 13.2.6). Otherwise the action will be as though comment had been written.

7.1.8. The end of a program.

The program extends from the first begin found in the input to the end which syntactically matches that begin. No further characters are input by the translator.

7.2. USE OF comment.

Following the delimiter comment not only any sequence of basic symbols, but any sequence of characters specified in section 6.5 is admitted up to the first following semicolon (;). For a special use of comments following end, see section 11.4.

7.3. ARITHMETIC VALUES AND OPERATIONS.

Integers are held in fixed-point form. This gives the range:
$-2\uparrow 39 = -549\ 755\ 813\ 888 < \text{integer} < 549\ 755\ 813\ 887 = 2\uparrow 39 - 1$

Reals are held as floating-point numbers. This gives the range of non-zero real values:

$2\uparrow(-512) = 7.458_{D}-155 < \text{abs(real)} < 1.341_{D}154 = 2\uparrow 512$

The precision of real values corresponds to 29 significant binary digits. Thus one unit in the last binary place will correspond to a relative change of the number of between $2_{D}-9$ and $4_{D}-9$.

As described below, the translator will sometimes insert an operation for converting arithmetic values from integer to real form, i.e. the float operation. This conversion is done by the instruction nkf39. This implies that integers larger than $2\uparrow 29$ in absolute value will have their rightmost digits set to zero.

When the result of an operation exceeds the number range, the system will sometimes call an alarm. This causes a print out of a message and a termination of the execution of the program.

In the following explanations of the arithmetic operations, the identifiers i and r stand for operands of types integer and real, respectively. Mixed type indicates one integer and one real operand.

The action of all arithmetic operators is given in the following table:

Operand types	Machine instructions	Reaction on range exceeded
	Addition, subtraction + -	
Integer	Fixed point	Incorrect result
Mixed	Float the integer, then as real	
Real	Floating	Spill alarm
	Multiplication ×	
Integer	Fixed point	Mult alarm
Mixed	Float the integer, then as real	
Real	Floating	Spill alarm
	Division /	
Integer	Float both, then as real	
Mixed	Float the integer, then as real	
Real	Floating	Spill alarm
	Integer divide, modulo : mod	
Integer	Special fixed point routines	Incorrect result
	Power ↑, yielding always a real result	
i ↑ i	Float left operand, then as r ↑ i	
r ↑ i	Successive floating point multiplications	See note below
i ↑ r	Float left operand, then as r ↑ r	
r ↑ r	Calculated as $2 \uparrow (r \times \log_2 r)$	Spill alarm

The power operator with an integer exponent calls the spill alarm on range exceeded. This happens already when the result of the operation abs(left operand ↑ (nearest greater power of two(abs(right operand)))) exceeds 2↑512.

In the evaluation of expressions involving several operations any necessary floating operations are done as late as possible consistent with the evaluation rules of Algol 60. For example, in the expression r + i × i the multiplication is done in integer form and only the result of it is floated before the addition. If it is desired to operate on integers in real mode arithmetics, one of the operands must be assigned to a real variables before the operation.

Round-off from type real to type integer is performed by means of the machine instruction tkf -29. This implies that real results of absolute value in the range from 0 to 2↑29 will yield correct integers on rounding, while reals with absolute value in the range from 2↑29 to 2↑39 will be rounded to an integer having too few significant figures. Real results larger than 2↑39 or smaller than -2↑39 will yield completely erroneous results if rounded.

7.4. RESERVED IDENTIFIERS.

A reserved identifier is one which may be used in a program for a standard purpose without having been declared in the program. If the standard meaning is not needed in a program the identifier may freely be declared to have other meanings.

The complete list of reserved identifiers arranged alphabetically is as follows:

Identifier	Reference
abs	3.2.4
arctan	3.2.4, 7.5.1
cancel	11.9
char	10.7
checksum	9
cos	3.2.4, 7.5.1
entier	3.2.5
exp	3.2.4
get	11.10
gier	12.7
il	11.13
kbon	10.2
ln	3.2.4, 7.5
lyn	10.4
put	11.10
readgeneral	10.12
readinteger	10.10
readreal	10.11
read string	10.13
reserve	11.7
select	8.1
sign	3.2.4
sin	3.2.4, 7.5.1
sqrt	3.2.4, 7.5.1
system	8.2
trackstransferred	11.3
us	11.13
where	11.8
write	9.7
writechar	9.2
writecr	9.3
writeinteger	9.6
writetext	9.4

7.5. STANDARD FUNCTIONS.

7.5.1. Precision.

The algorithms for calculating the standard functions arctan, cos, exp, ln, sin, and sqrt, incorporated in Gier Algol will all yield results having an error less than that which corresponds to about 2 units in the last place of the result or the argument, whichever gives the greater error. As one consequence of this the functions sin and cos for absolute values of the arguments larger than about $_{10}10$ are worthless, their values being usually 0 or 1.

7.5.2. Alarms.

Certain misuses of the standard functions will cause termination of execution of program (see Appendix 2). Note, however, that ln(0) will supply the result $-9.35_{10}49$ and not call the alarm.

7.6. ARITHMETIC EXPRESSIONS.

The treatment of arithmetic types and the precision of real arithmetic are described in section 7.3. Alarms are described in Appendix 2.

Several additional operators have been included. abs, entier, round, integer, and real are monadic operators of high precedence, which may precede any operand and one another in an arithmetic expression, thus for example:

```
f := abs r + sin(real b) - (if -, real b < integer h then entier c else
     round d↑abs f) - abs real m;
```

The precedence of these new operators being higher than that of any other, this expression will be evaluated as if the following form had been written:

```
f := (abs r) + sin(real b) - (if -,(real b) < (integer h) then (entier c)
     else (round d)↑(abs f)) - (abs(real m));
```

Their effect is as follows:

Operator	Operand type	Result type	Effect
abs	integer or real	integer or real	Absolute value, like the standard procedure abs, but without change of type
entier	real	integer	Greatest integer not greater than the value of the operand
round	real	integer	Nearest integer
integer	real or boolean or string	integer	Integer represented internally by the same bits as the operand
real	integer boolean or string		Real represented in internal, floating, packed form by the same bits as the operand

The operator mod (modulo) resembles : (integer divide) in that it is dyadic, has the same precedence as :, and requires two integer operands. The result is the remainder of the integer division:

a mod b is the same as a - (a : b) × b

provided that a and b produce no side-effects.

7.7. BOOLEAN EXPRESSIONS, BIT PATTERNS, AND STRINGS.

7.7.1. Boolean values.

Boolean variables serve both as single binary values and as patterns of 40 bits. Used as bit patterns and supplemented with the operators integer, real, boolean, and string, they will allow arbitrary manipulations of parts of machine words.

The positions of a bit pattern are numbered from 0 (left) to 39 (right). The truth value of a bit pattern is given by the value of the bit in position 0 as follows:

0 corresponds to false
1 - - true

Literal bit patterns may be written directly in the Algol program in three different ways:

1) true has bit value 1 and false has bit value 0, in all positions 0 to 39.

Note that boolean values computed by means of relational operators generally will not be represented by the bit patterns written in the program as true and false.

2) Arbitrary patterns may be written in a notation which builds them up from part patterns as follows. Each part pattern is written as one underlined, unsigned integer, indicating the number of bits in the part pattern, followed either by an unsigned integer, giving by its binary representation, the actual pattern, or by the letter m, denoting a part pattern having digit 1 in all positions. According to the taste of the reader the letter m may be thought of as a picture of several ones, or to stand for the word many. Examples of part patterns:

Notation in program	Pattern
4 5	0101
6 33	100001
11 1022	01111111110
5 m	11111

Parts patterns may be joined to one another by writing them next to one another. The final 40 bit pattern is obtained by filling in extra 0-bits to the right, as necessary. Example: the pattern written

4 5 6 33 11 1022 5 m

will look as follows (where to help the human reader the 40 bits have been grouped by fives):

01011 00001 01111 11111 01111 10000 00000 00000

3) Bit patterns may be written as digit layouts, as used to control certain output procedures (cf. section 9.5).

The logical operators, =, =>, ∨, ∧, and -, will operate on all 40 bits of their operands in parallel, and may thus be used to manipulate bit patterns. Examples of applications:

v ∧ 12 7

extracts from v the bits in positions 9, 10, and 11, setting the remainder of the pattern to 0.

w ∨ y

packs the bits of w and y into one pattern.

As a further aid to the effective utilization of bit patterns an operator, shift, has been added. shift is a dyadic operator of low precedence, requiring as its first operand a boolean value, which is interpreted as a bit pattern, and as its second operand an integer value, interpreted as a number of cyclic shifts of the pattern. The result of the operation is of type boolean and is the bit pattern obtained from the first operand by performing that number of cyclic shifts which is given by the second operand, shifting left for a positive number, right for a negative number. Examples:

p = q shift j + k

By the low precedence of shift this is evaluated as

(p = q) shift (j + k)

Suppose we have executed

w := 20 13 10 9;

Then the result of

w shift -5

can be written 25 13 10 9. The result of w shift -15 is 5 9 30 13. The number of shifts should be kept between -512 and 511, but there is no check of this.

7.7.2. String values and expressions.

Within a string is admitted, not only any sequence of basic symbols, but any sequence of characters, with the exceptions mentioned in section 7.1.3.2.

In the internal representation the characters of strings are packed into one or more machine words omitting the string quotes. Each character uses 6 bit positions, corresponding to the value of the character given in section 6.5, with the exceptions:

CAR RET is represented as 63
, - - - 0, i.e. as BLANK

Characters for UPPER CASE and LOWER CASE are included as needed, but all strings are understood to begin and end in lower case. The end of a string is indicated by the character value 10. Strings having 6 or fewer characters are packed into one word and appear in this way at run time. Longer strings are stored on the backing store like program segments and appear as words referring to the backing store.

Pattern representation of short string (6 or fewer characters):

Bits 0 - 3	Pattern 1010, i.e. decimal value 10	
- 4 - 9	Character no. 6	
- 10 - 15	- - 5	Unused character positions are set to 001010, i.e. decimal value 10
- 16 - 21	- - 4	
- 22 - 27	- - 3	
- 28 - 33	- - 2	
- 34 - 39	- - 1	

The word referring to a long string has zeroes in bit positions 0 - 9 and 20 - 29. The remaining positions supply the track number and track relative address. However, since the track number is counted relative to a base number which is not directly available to the user, not even using machine code, the pattern form will not be given.

For additional facilities related to strings, see section 9.4.4.

In addition to string values, more general string expressions may be formed as a conditional or case expression and by using the operator string. An example of a string expression is as follows:

if b then ⟨<constant⟩ else string (p-q)

String expressions should be added to the class of expressions of Algol 60.

The internal bit pattern representation of values of types integer, real, and string, may be obtained with the aid of the operator boolean, which like the operators integer, real, and string, has high precedence, and has as result a boolean value which is the bit pattern used internally for representing the operand.

As a general rule, the operators integer, real, boolean, and string, do not give rise to any action during program execution. They merely suppress the type alarm action which would take place during translation if they were omitted.

7.8. INTEGERS AS LABELS.

Integers cannot be used with the meaning of labels in Gier Algol.

7.9. FOR STATEMENTS.

The controlled variable must be simple.

7.10. PROCEDURE STATEMENTS.

7.10.1. Recursive procedures.

Recursive procedures will be processed fully in Gier Algol.

7.10.2. Handling of types.

Gier Algol 4 checks that the actual parameters of a procedure matches the corresponding specifications whenever the identifier of the procedure is not formal. In parameters called by name, strict type agreement is required, both for simple variables, expressions, array and procedure identifiers, integer specification requiring an actual parameter of integer type, real specifications one of real type, etc. In parameters called by value, integer specifications may correspond to real parameter and real specification to integer parameter.

In a call where the procedure identifier is formal, no check of the actual parameters is made. If the parameters do not match the formals as indicated above the result is unpredictable.

7.10.3. Standard procedures.

A number of special actions, including input and output of data, are expressed as calls of standard procedures. These calls conform to the syntax of calls of declared procedures (cf. section 4.7.1) and should be regarded in all respects as regular procedure calls or funtion designators, with the exception that identifiers of standard procedures with parameters may not be used as actual parameters.

7.11. ORDER OF DECLARATIONS.

In Gier Algol declarations may appear in any order in the block head.

7.12. Own.

In Gier Algol own can only be used with type declarations, not with array declarations.

7.13. PROCEDURE DECLARATIONS.

7.13.1. Recursive procedures.

Recursive procedures will be processed fully in Gier Algol.

7.13.2. Arrays called by value.

Gier Algol cannot handle arrays called by value.

7.13.3. Specifications.
The specifications of formal parameters must be complete, i.e. each parameter must occur just once in the specification part.

7.13.4. Labels called by value.
Labels cannot be called by value in Gier Algol (the Revised Algol 60 Report leaves the question unanswered).

7.14. CASE EXPRESSIONS.

Gier Algol 4 includes an extension, known as case expressions and statements, which was first proposed by C. A. R. Hoare in the IFIP Working Group 2.1 on Algol. These constructions are a natural generalization of conditional expressions and statements, for expressing a choice, not only between two but among any number of possibilities.

7.14.1. Syntax.
<case expression> ::= <case clause>(<expression list>)
<case clause> ::= case <arithmetic expression> of
<expression list> ::= <expression>|<expression list>, <expression>
The complete case expression may be written whereever an expression of the same type is admitted.

7.14.2. Examples.
case k of (a, b-c, d, g)
case round q of (a/b, if c = d then w else t, case s of (p,q,y))

7.14.3. Semantics.
A case expression is evaluated as follows. First, evaluate the arithmetic expression of the case clause. Next, select that expression of the expression list which corresponds to the result of the first evaluation, in the sense that the result 1 corresponds to the first expression, the result 2 to the second, etc. Finally, evaluate the expression thus selected to obtain the value of the complete case expression.
The arithmetic expression of the case clause must be of type integer. If there exists no expression corresponding to its value, the execution will terminate with an alarm. The types of the individual expressions of the expression list must be compatible. The type of the complete case expression is the same as that of the constituents if these are all alike, and real if both types integer and real occur.

7.14.4. Limitation.
The expression list may contain at most 34 expressions.

7.15. CASE STATEMENTS.

7.15.1. Syntax.
<case statement> ::= <case clause> begin <statement list> end
<statement list> ::= <statement>|<statement list>; <statement>
The complete case statement may be written whereever a statement is admitted.

7.15.2. Examples.

```
case k of begin q := s; m := t - y; go to v end
case t - u of begin P(u);
                    case v of begin t := y; y := w := p; u := r end;
                    s := r - b
               end
```

7.15.3. Semantics.

A case statement is executed as follows. First the arithmetic expression of the case clause is evaluated and thereby selects one of the statements of the statement list. Next, this statement is executed. Unless this execution defines its successor explicitly, the execution then continues with the statement following the complete case statement.

If there exists no statement corresponding to the value of the expression of the case clause, the execution terminates with an alarm.

7.15.4. Limitation.

The statement list may contain at most 34 statements.

7.16. GENERAL LIMITATIONS.

Gier Algol imposes a number of limitations caused by the finite size of the tables used during compilation. However, with one exception these limitations shall not be mentioned further here, partly because only very exceptional programs are likely to exceed the capacity, partly because alarm messages during compilation will indicate when they are violated (see appendix 3). The exception is the limitation that the number of variables which are active simultaneously at any time during the execution of a program must be confined to about 650. This problem is discussed in detail in section 11.2.

In machines equipped with the additional buffer store unit, the elements of arrays may be stored there, and the capacity will be about 4000 subscripted and 650 simple variables.

8. EXTERNAL MEDIA AND MACHINE CONFIGURATIONS.

8.1. STANDARD PROCEDURE: select.

At run time the selection of the data input and output medium is controlled partly by select-statements in the Algol text, partly by a mask which may only be controlled by actions outside the realm of the Algol text, in a manner which will not be described further here.

The select-statements correspond to a standard procedure with the following heading:

integer procedure select(u); integer u;

The action of select is associated with a machine instruction held in the run-time administration, of the following form:

vy last select t mask

where mask is 896 unless it has been changed by an action outside the Algol program. The action of select may be described as follows:

select := last select;
last select := u;
execute the vy instruction;

The execution of the vy instruction assigns the value of last select, properly masked, to the by-register. There is no check that the value of u is sensible.

Two consequences of this action may be noted. First, if the mask is unchanged, a modification of the contents of the by-register by direct operator action, using the push-buttons of the control panel of the machine, will remain active until the next following call of select, while it is not influenced by input or output statements. Second, if the mask is changed, such modifications of the contents of the by-register may be made insensitive to calls of select.

The significance of the bits of the by-register depends on the peripheral units attached to the machine and therefore differs somewhat from one machine to the other. A common arrangement is as follows:

by-value	Meaning
0	Input from paper tape reader with stop on parity errors
1	Input from typewriter
3	Input from paper tape reader, no stop on parity errors
8	Output on line printer
16	Output on typewriter
32	Output on paper tape punch

It is possible to select one input medium and several output media in one operation, by calling select with the sum of the corresponding by-values. Thus, for example, in order to select input from paper tape reader and output on typewriter and on punch, we must call select(51).

At first entry into the Algol program, last select and the by-register are normally set to 35: input from paper tape, output to punch.

In general, the by-values appropriate to the particular machine in which the program is operating may be obtained through a call of system (cf. 8.2). In this way it is possible to write programs which accomodate themselves in the particular machine environment. In programs written only for one machine, absolute by-values may of course be used without difficulty.

8.2. STANDARD PROCEDURE: system.

8.2.1. Implied procedure heading

procedure system(A); <type> array A;

8.2.2. Semantics.

Each call of system transfers an array of 40 elements, describing the characteristics of the surrounding machine, to the array given as parameter. The first version of the form of this description is given in appendix 4. Because of the possibility of so far unforseen extensions of machines, this description may not remain complete in the future. The form has been chosen in such a way that it leaves a considerable capacity open for future extensions, without thereby making it necessary to modify the existing conventions.

The array given as parameter must have precisely 40 elements, otherwise an execution alarm is called.

9. STANDARD OUTPUT PROCEDURES.

The standard output procedures serve to transfer the results of programs to external media. Upon transfer, the results must always exist in the form of strings of characters. These characters and their corresponding internal, integer values are given in section 6.

It is common to all standard output procedures that the medium to which output is made is controlled by calls of standard procedure select (section 8.1). Another common feature is that a check sum of the output characters is formed. This check sum is accessible to the programmer through a standard procedure of the following description:

```
integer procedure check sum(u); integer u;
    begin check sum := character sum; character sum := u end;
```

This check sum is of limited utility, however, since it may be checked only if the external medium is read by means of standard procedure lyn, but not by any other input procedure or by the translator (cf. 7.1.4).

9.1. IDENTIFIERS AND MAIN CHARACTERISTICS.

Identifier	Example, reference	Effect
writechar	writechar(49) section 9.2	procedure writechar outputs the character corresponding to the value of the parameter.
writecr	writecr section 9.3	procedure writecr outputs one CAR RET character.
writetext	writetext(‹‹FI=›) section 9.4	procedure writetext outputs a string of symbols.
writeinteger	writeinteger(‹pdd.dd›, n) section 9.6	procedure write integer outputs a value given as an integer, but with a decimal point inserted in a specified location.
write	write(‹-dd.dd›, q) section 9.7	procedure write outputs the values of an arbitrary number of arithmetic expressions in a specified digit layout.

9.2. STANDARD PROCEDURE: writechar.

9.2.1. Implied procedure heading

```
procedure writechar(u); integer u;
```

9.2.2. Examples.

```
writechar(49)
writechar(symbol - case)
```

9.2.3. Semantics.

Each call of writechar causes the character corresponding to the value of the actual parameter to be output. The correspondence between the integers and the characters is given in section 6.5. If the value of the actual parameter is negative or larger than 127, the effect is undefined.

The UPPER CASE and LOWER CASE characters must be output explicitly where needed. Where writechar is called side by side with calls of writetext, writeinteger, or write, it is important to note that these latter will assume the output to be in lower case when a call is made and will also leave it in lower case when the call is completed.

9.3. STANDARD PROCEDURE: writecr.

9.3.1. Implied procedure declaration

```
procedure writecr; writechar(64);
```

9.3.2. Example

```
writecr;
```

9.3.3. Semantics

The effect of the call is fully explained in section 9.3.1.

9.4. STANDARD PROCEDURE: writetext.

9.4.1. Implied procedure heading

```
procedure writetext(u); string u;
```

9.4.2. Examples

```
writetext(‡<alpha = ‡);
writetext(formal string);
writetext(string 4 10 6 10 6 35 6 38 6 55 6 35 6 49);
writetext(if q then ‡<yes‡ else ‡<no‡);
```

9.4.3. Semantics

Each call of writetext causes output of the characters of the proper string resulting from the evaluation of the actual parameter. String values and expressions are described in section 7.7.2.

9.4.4. Variable strings.

Short strings may be manipulated as bit patterns in the form given in section 7.7.2. When used as parameter to writetext the pattern must be converted to string type with the aid of the operator string, see section 7.7.

In order to allow also manipulation of long strings, the action of procedure writetext is as follows: Whenever the procedure finds the character value 15 in the string word, this is taken as a signal that a new string word should be obtained through a repeated call of the actual parameter. To make use of this action, the programmer must write the actual parameter so that a series of calls of it will deliver the successive parts of the string to be output. Each part of the string must have the format of a short string, as given in section 7.7.2, except that bits

0 - 3 must have the pattern 1111, i.e. decimal value 15. The output terminates when the character value pattern 1010, i.e. decimal value 10, is found.

As an illustration the following block causes the string packed in character form into the successive elements of Boolean array TEXT[0:q] to be output.

```
begin integer p;
integer procedure p step;
    begin p step := p; p := p + 1 end;
p := 0;
write text(string TEXT[p step])
end;
```

The packing of characters into TEXT must conform to the following general pattern:

	Bits 0-3	4-9	10-15	16-21	22-27	28-33	34-39
TEXT[0]	15	no.6	no.5	no.4	no.3	no.2	no.1
TEXT[1]	15	no.12	no.11	no.10	no.9	no.8	no.7
...				.....			
TEXT[m]	10	Last characters, use 10 as filler					

If by mistake the leading bits are neither 10 nor 15, the procedure will output the corresponding character and exit.

9.5. LAYOUTS.

The standard procedures writeinteger and write, described in sections 9.6 and 9.7, use bit patterns to control the form of the character string representation of numbers. The bit patterns are values of type Boolean, as discussed in section 7.7. For use with the output procedures these values are usually most conveniently written in the form of layouts, as described below.

9.5.1. Syntax

```
<sign> ::= <empty>| - | + | +
<exponent layout> ::= ₁₀<sign>d|<exponent layout>d
<zeroes> ::= 0 |<zeroes>0 | <zeroes>␣0
<positions> ::= d | <positions>d | <positions>␣d
<0-positions> ::= <positions> | <0-positions>0 | <0-positions>␣0
<p-positions> ::= p|<positions>|p<positions>|p␣<positions>
<p-0-positions> ::= <p-positions>|<p-0-positions>0|<p-0-positions>␣0
<decimal layout> ::= <p-0-positions>|<p-0-positions>.<zeroes>|
    <p-positions>.<0-positions>|.<0-positions>
<layout tail> ::= <decimal layout>|<decimal layout><exponent layout>
<layout string> ::= <sign><layout tail>|␣<layout string>
<layout> ::= {<layout string>}
```

In this syntax BLANK and ␣ may be used interchangeably.

9.5.2. Examples

```
<,,dd,dd>
<-pddd00.0,0_10+dd>
<+p>
<,,,p,dd.dd0,0>
<,,,.dd>
```

9.5.3. Pattern representation of layouts

The 40-bit pattern representation of a layout may be derived from the following rules:

Bits	
0 - 19	Representation of BLANKs: First, a 1 for each leading BLANK of layout. Second, one 0. The following positions correspond to the following character positions of the decimal layout with the sign and BLANKs omitted. The pattern has 1 if the corresponding layout position is followed by BLANK, otherwise 0.
20 - 23	b = number of significant digits, i.e. p and d 4
24 - 27	h = - - digits before the point 4
28 - 29	fn = sign of number part (no sign = 0, - = 1, + = 2, $\pm$ = 3) 1
30 - 33	d = number of digits after the point 4
34	p, 0 if not present, 1 if present 1
35 - 37	s = number of digits in exponent 2
38 - 39	fe = sign of exponent (coded as fn) 1

In the following example, BLANKs in the bit pattern have no significance other than to help the reader:

```
                    h
                 ┌──────┐
                   b        d     s
                 ┌─────┐  ┌───┐  ┌┐
Layout string:   ,,+pdd,dd0.00,0_10-dd
Pattern:         110001 000001 00000000 0101 0110 10 0011 1 010 01
                                         b    h   fn  d   p  s  fe
```

9.5.4. Limitations.

Only such layout strings which may be represented by bit patterns, as shown in section 9.5.3, are acceptable to the translator. Consequently, the following limitations must be observed:

Within the decimal layout: The total number of symbols p and d must be ≤ 15; the total number of symbols p, d, and 0, written to the left of the point must be ≤ 15; the total number of symbols d and 0 written to the right of the point must be ≤ 15; the sum of the number of leading BLANKs and the number of character positions from the first non-BLANK to the rightmost character position preceding a BLANK, not counting BLANKs, must be ≤ 19.

The number of symbols d in the exponent layout must be ≤ 7.

9.6. STANDARD PROCEDURE: write integer.

9.6.1. Implied procedure heading
procedure write integer(LAYOUT, EXPRESSION);
value LAYOUT; Boolean LAYOUT; integer EXPRESSION;

9.6.2. Examples
writeinteger(⟨-ddd.dd⟩, q - t)
write integer(q ∧ (v shift 16), K[i])

9.6.3. Semantics

Each call of write integer causes output of the value of the expression given as second parameter in a form controlled by the first parameter. In order to achieve high speed, only some of the features of the layout string (or, equivalently, the bit pattern) have an effect on the form of output thus:

(1) Exponent layouts have no effect and should be omitted.
(2) Zeroes should not be used.
(3) Following the leading BLANKs the layout string should continue with either p or - and no p. Thus the signs + and ± cannot be used.

The layout should be regarded as a picture of the final output character string, which will have one position for each position written in the layout string. The digits of the value of the EXPRESSION will be placed in the p and d positions, aligned such that units will be placed in the rightmost position of the layout string. BLANKs and point will be inserted where they occur in the layout string. If the layout string contains - (minus) the first position to the left of the first digit printed will be printed as - if the value of the EXPRESSION is negative, otherwise as BLANK. The treatment of leading zeroes of the numeral, i.e. of the digit positions to the left of the first digit which is different from zero, is controlled by the presence of p and point in the layout string. If p is present, the leftmost zeroes are always output as zeroes. If p is not present, leftmost zeroes until, but not including, the first position to the left of the point are output as BLANK, while any following zeroes are output as zero.

9.6.4. Alarm printing

By alarm printing is meant that the value of the EXPRESSION cannot be accomodated in the form described by the layout string. If the value is too large the output will correspond to a layout obtained by adding the sufficient number of d-s to the left of the point or first d of the layout string.

9.6.5. Limitations and other possibilities

The procedure will never output a numeral of more than 12 characters. If the layout has more than 12 p-s and d-s, the effect will be as though some of the d-s had not been present.

If the layout does not satisfy the above mentioned limitations some output will be produced, in a form which will not be described in full here. We only mention that if a negative number is output with a layout without a minus sign, the output has the wrong sign.

9.6.6. Examples of layouts and output

In order to indicate the exact output, commas are inserted immediately preceding and following each numeral.

Number	⟨pdd,ddd⟩	⟨,,,pdd.dd,dd⟩
0	,000 000,	, 000.00 00,
1	,000 001,	, 000.00 01,
23456	,023 456,	, 002.34 56,
333444555	,333 444555,,	, 333.44 4555,

Number	⟨-dd,ddd⟩	⟨,,,-.ddd,dd⟩	⟨,,-dd.dd⟩
0	, 0,	, .000 00,	, 0.00,
1	, 1,	, .000 01,	, 0.01,
-234	, -234,	, -.002 34,	, -2.34,

9.7. STANDARD PROCEDURE: write.

9.7.1. Implied procedure heading

procedure write(LAYOUT, EXPR1, EXPR2, ...);
value LAYOUT, EXPR1, EXPR2, .. ;
Boolean LAYOUT; real EXPR1, EXPR2, ... ;
The call may have any number of expressions as parameters.

9.7.2. Examples.

write(⟨ddd.00⟩, P)
write(⟨,,,-d$_{10}$-dd⟩, eps, delta, q/16)
write(lay0 ∧ (m shift 16) ∧ (m shift 12), p - q)

9.7.3. Semantics

Each call of write causes output of the numerals representing the values of the second and following parameters, in a form defined by the value of LAYOUT. In what follows this form is described in terms of the layout string. If the LAYOUT is given as a pattern, the form of output is defined through the correspondence between a layout and a pattern given in section 9.5.

The layout string gives a symbolic representation of digits, blanks, and special characters of the numeral produced as output. Indeed, except for alarm printing, the numeral will have exactly the same number of characters as is present in the layout. The symbols of the layout have the following significance:

9.7.3.1. Sign. The four possible signs signify:

Empty. The number is supposed to be positive. No sign will be output. A negative number causes alarm printing (section 9.7.4).

- (minus). A sign will be output, using BLANK for positive, and - for negative numbers. It will appear as the first or second character to the left of the first digit or the decimal point, with at most a layout BLANK in between.

+ (plus). A sign will be output, using + for positive, and - for negative numbers, placed as explained for - (minus).

+ (plus minus). Using + for positive and - for negative numbers, the sign will be output as the first character following leading BLANKs.

9.7.3.2. Digits. Letters d and p represent digits. Letter p may only appear as the first character following the sign. The total number of letters d and p gives the maximum number of significant digits in the numeral. Small numbers will appear with less than this number of significant digits. If p is used all leading zeroes of the numeral will be output as 0. Otherwise such leading zeroes will be printed as 0 only in the units' position and in positions to the right of the decimal point, while in positions to the left of the units' they will be output as BLANK.

9.7.3.3. Zeroes. Zeroes may appear at the end of a decimal layout. They influence the representation of the number in the following manner. If m zeroes are present at the end of the decimal layout, the exponent output will be exactly divisible by m+1. For this to be possible at the same time as the position of the decimal point within the complete numeral is kept fixed, the significant digits are allowed to move to the right, using the zero-positions, to an extent depending on the magnitude of the number. If no exponent layout is included the exponent 0 is understood and the rule holds unchanged.

9.7.3.4. BLANKs. BLANK will appear in the numeral in all positions where the layout string has the character BLANK or _.

9.7.3.5. Decimal point. The decimal point will always be printed in a fixed position within the numeral. If the numeral includes digits to the right of the point, it will appear as . otherwise as BLANK.

9.7.3.6. Scale factor. An exponent layout will give rise to output of a scale factor in the numeral. The character 10 will appear immediately before the exponent sign. If the scale factor is unity, the whole scale factor will be replaced by BLANKs. Note that the layout string cannot contain an exponent layout without a decimal layout.

9.7.3.7. Round off. Before output, all numbers will be correctly rounded off to the number of significant digits given in the numeral.

9.7.4. Alarm printing.

By alarm printing is meant that the numeral will have more positions than the layout string. Alarm printing will occur as follows:

9.7.4.1. Negative number output with layout having empty sign. The correct - will be inserted, using one extra position.

9.7.4.2. Number too large for layout. In this case the layout actually used is derived from the one given as parameter by inserting an exponent layout, or by increasing the number of exponent digits.

9.7.5. Examples of layouts and output

In order to indicate the exact output, commas are inserted immediately preceding and following each numeral.

‹ p ddd ddd›	‹ ±d ddd.dd›	‹ +d ddd.ddd d›	‹pdd ddd›
, 0 000 000,	, + 0.00,	, +0.000 0,	,000 000,
, 0 000 000,	, + 0.00,	, +0.001 2,	,000 000,
, 0 000 000,	, + 0.12,	, +0.123 5,	,000 000,
, 0 000 001,	, + 1.23,	, +1.234 6,	,000 001,
, 0 000 012,	, + 12.35,	, +12.345 7,	,000 012,
, 0 000 123,	, + 123.46,	, + 123.456 8,	,000 123,
, 0 001 235,	, +1 234.57,	, +1 234.567 9,	,001 235,
, 1 234 570₁₀3,	, +1 234.57₁₀6,	, +1 234.567 9₁₀6,	,123 457₁₀4,
, -0 012 346,	, -1 234.57₁₀1,	, -1 234.567 9₁₀1,	,-012 346,

‹-ddd.d00₁₀+d›	‹ddd.d0000₁₀ddd›	‹+pdd.dd₁₀-d›	‹±dddd00₁₀±d›
, 0.000 ,	, 0.00000 ,	,+000.00 ,	, 0 ,
, 1.235₁₀-3,	,123.5 ₁₀-5,	,+123.46₁₀-5,	,+ 1235₁₀-6,
, 123.5 ₁₀-3,	, 0.1235 ,	,+123.46₁₀-3,	,+123500₁₀-6,
, 1.235 ,	, 1.235 ,	,+123.46₁₀-2,	,+ 1235₁₀-3,
, 12.35 ,	, 12.35 ,	,+123.46₁₀-1,	,+ 12350₁₀-3,
, 123.5 ,	,123.5 ,	,+123.46 ,	,+123500₁₀-3,
, 1.235₁₀+3,	, 0.01235 ₁₀5,	,+123.46₁₀ 1,	,+ 1235 ,
, 1.235₁₀+9,	, 0.1235 ₁₀10,	,+123.46₁₀ 7,	,+ 1235₁₀+6,
, -12.35 ₁₀+3,	, -0.1235 ₁₀5,	,-123.46₁₀ 2,	,- 12350 ,
, 12.35 ₁₀+15,	, 12.35 ₁₀15,	,+123.46₁₀ 14,	,+ 12350₁₀+12,

10. STANDARD INPUT PROCEDURES.

For input of data from external media, the system provides six different procedures. One of them, kbon, inputs a single bit, another, lyn, reads one single character. Three others, read integer, read real, and read general, are numeral-reading procedures, i.e. they include conversion of numerals, i.e. character strings containing digits and other special characters and representing numbers, to the internal representation of numbers. The last of the six, read string, reads the input as a string of characters. The set of numeral-reading procedures and read string we shall refer to as the bulk reading procedures.

10.1. IDENTIFIERS AND MAIN CHARACTERISTICS.

Identifier	Example, reference	Effect
kbon	if kbon then Section 10.2	Boolean procedure kbon supplies the current value of the manually operated KB register.
lyn	k := lyn Section 10.4	integer procedure lyn reads the next character from the input medium
char	char := 32 Section 10.7	integer char, a standard variable, will be used as the first input character at every call of read integer, read real, read general, and read string. At completion of the call of either of these procedures it contains the last character input from the medium.
read integer	p := read integer Section 10.10	integer procedure read integer has the next integer numeral appearing on the input medium as its value.
read real	r := read real Section 10.11	real procedure read real has the next real numeral appearing on the input medium as its value.
read general	read general(A,b,n) Section 10.12	integer procedure read general inputs a series of numerals and delimiters and assigns their value to successive elements of an array.
read string	read string(B,q,n) Section 10.13	integer procedure read string inputs a string of characters and packs it into successive elements of an array.

10.2. STANDARD PROCEDURE: kbon.

10.2.1. Implied procedure heading

Boolean procedure kbon;

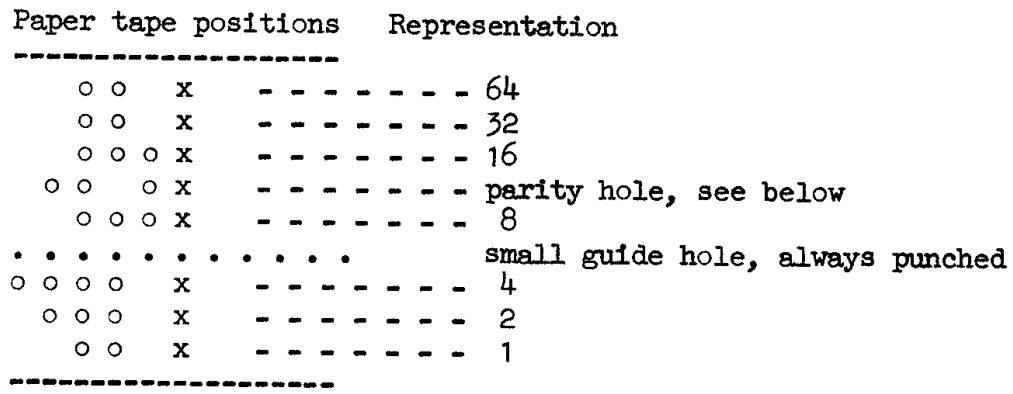
10.2.2. Example
if kbon then go to PRINT

10.2.3. Semantics
The value of kbon is given by the current state of the manually controlled KB register of the machine; it is true when KB is on, otherwise false.

10.3. INPUT MEDIUM AND CHARACTER REPRESENTATION.

The input medium activated by the procedures lyn, read integer, read real, read general, and read string, is controlled with the aid of the standard procedure select (section 8.1).

These same five standard procedures communicate characters. Internally, characters are represented by integers. The representation of printing characters received from external media is given in section 6.5. More generally, the integer representation of a hole combination may be obtained by first adding the integers corresponding to those hole positions which are punched, according to the following picture:



To this sum must be added 512 if the hole pattern has even parity, i.e. if it has an even number of holes. The representations of the five patterns given as examples are: 4, 6, 639, 127, and 24. On some machines the value of ALL HOLES comes out as 127, however (see appendix 4).

10.4. STANDARD PROCEDURE: lyn.

10.4.1. Implied procedure heading
integer procedure lyn;

10.4.2. Examples
```
symbol := lyn
w := table[lyn + case]
```

10.4.3. Semantics

Each call of lyn causes the next character in the external medium to be read and yields the internal representation of the character as result. A call of lyn has no influence on the value of char, as described in section 10.7.

10.5. LOWER AND UPPER CASE.

Before it is used by one of the numeral-reading procedures, a character is often combined with the last LOWER CASE or UPPER CASE character encountered in the input string, to form a character-with-case. This is obvious in the case of printing characters which appear differently in the two cases, as shown in section 6.5. Where it is not obvious we shall in the following indicate the treatment of case explicitly.

10.6. BLIND CHARACTERS.

The numeral-reading procedures completely ignore the following characters whenever they appear in the input, irrespective of the case:

Name	Internal representation	Hole pattern
BLANK	0	, o . ,
TAPE FEED	63	, oooo.ooo,
ALL HOLES	127 or 639	,ooooo.ooo,
DUMMY	127	,ooo o.ooo,

The same holds for read string, except that it does not ignore BLANK.

10.7. STANDARD INTEGER: char.

The standard integer char serves to make the last character read in any call of one of the bulk reading procedures available to the user. Thus, any of these procedures will assign the value corresponding to the last character read from the external medium to it. In addition, every call of one of these procedures will use the value of char as the first character read in, before any characters supplied from the external medium. Thus, if the value of char is not otherwise changed, the last character read by one of these procedures will be read once more by the next bulk reading procedure called.

Apart from this coupling to the bulk reading procedures, char may be regarded as a simple variable of type integer. Thus it may be assigned to, as for example:

```
char := 49
```

and its current value may be used in expressions.

The value of char assigned by a bulk reading procedure includes the case as follows:

```
value of char = representation of character +
      (if lower case then 0 else 128)
```

The value of char used as the first character of input is obtained from the actual value by replacing the first 30 bits of its binary representa-

tion by zeros. Before every execution of a program, char is initialized to 0.

The case indication contained in the value of char when used by a bulk reading procedure, will act like a case character read from the input medium. It is therefore possible that the following characters from the input medium will not be read in their original case context. On the other hand, this feature will enable the user to take input from alternate media, without any risk that the case situation of any of them is lost. For this purpose the user must record the value of char before every selection of another medium and assign that value to char when the original medium is selected the next time. Another use of the feature is to keep the correct case when alternating between input by means of lyn and the bulk reading procedures.

10.8. EXIT CONDITIONS.

The bulk reading procedures will accept any character string from the input medium, without ever calling an alarm. Unless the input medium is exhausted, the procedures will return from the call with information about the character string read and about the conditions causing the exit.

10.9. NUMERAL RECOGNITION AND OVERFLOW.

The three numeral-reading procedures all conform to the following common rules about the recognition of a numeral in the input string:

1) Blind characters are ignored completely in all contexts.

2) To be recognized as a numeral, a section of the input string must contain at least one digit.

3) Of the characters preceding the first digit, either none, one, two, or three, will be regarded as part of the numeral, depending on the particular procedure and the context.

4) The numeral usually comprises a sequence of digits and other characters, according to syntactic rules given for each procedure. It is usually terminated by a non-digit in the input, which acts as terminator of the numeral. The further action taken on the terminator depends on the procedure.

5) Exceptional termination of a numeral occurs whenever the sequence of characters starting with leading digit, after removal of at most one decimal point, forms an integer numeral in excess of

$$2\uparrow 39 - 1 = 549\ 755\ 813\ 887$$

In this case the last incoming digit is said to cause overflow. The reaction of the procedures is to remove that last digit from the numeral and to treat the digit as the terminator.

10.10. STANDARD PROCEDURE: read integer.

10.10.1. Implied procedure heading

integer procedure read integer;

10.10.2. Examples
p := read integer;
s := q[read integer + 4];

10.10.3. Semantics

This procedure is accomodated on a single program segment and will read at very high speed. Each time it is called it will start reading from the external medium, according to the rules given in sections 10.3, 10.5, 10.6, 10.8, and 10.9 and yield the number corresponding to the next following numeral in the input as result. The syntax and interpretation of the numeral in the input is as follows:

1) If the first character preceding the first digit of the numeral is minus (internal representation 32) the number is negative, otherwise it is positive.

2) Excepting overflow, the numeral includes the following digits, up to the first following non-digit, which is the terminator. The numeral is interpreted as an ordinary decimal integer.

As an example, the results of reading the following input string:
pop - s 27, -43-888+555- 333.497 222 333 444 888 333;
by means of several calls of read integer, with no intervening assignments to char, are as follows:

	Value of read integer	char
First call	27	27
Second call	-43	32
Third call	-888	160
Fourth call	555	32
Fifth call	-333	59
Sixth call	497 222 333 444	8
Seventh call	888 333	133

10.11. STANDARD PROCEDURE: read real.

10.11.1. Implied procedure heading
real procedure read real;

10.11.2. Example
r := read real/33;

10.11.3. Semantics

Of the three numeral-reading procedures, only read real will recognize exponents as parts of numerals. It requires two program segments and is slower in operation than read integer. It operates like read integer, but the syntax and interpretation of the numeral in the input is as follows:

1) The first part of the numeral will contain a digit, which is interpreted according to rule 2 below. The reading continues as long as the characters encountered do not conflict with the syntax of the left end of a number in the sense of Algol 60. In general, the numeral will thus have digits in the integral, the fractional, and the exponent part, corresponding to before the decimal point, after the decimal point, and after the

exponent ten, and is interpreted according to the rules of Algol 60. Where the resulting numeral is an incomplete, left end of an Algol 60 number as for example 2.a, or 7₁₀-q, the value yielded by read real corresponds to the numeral obtained by inserting a zero at the right end, thus in the two examples: 2.0a and 7₁₀-0q. Examples of numerals in input and the numeral interpreted by read real are given below.

2) The interpretation of the first digit of the numeral depends on the preceding one, two, or three, characters, as follows:

Third preceding	Second preceding	First preceding	First digit belongs in:
-	₁₀	+ or -	exponent of negative number
not minus	₁₀	+ or -	exponent of positive number
anything	-	₁₀	exponent of negative number
anything	not minus	₁₀	exponent of positive number
anything	-	.	fractional of negative number
anything	not minus	.	fractional of positive number
anything	not ₁₀	-	integral of negative number
anything	anything	not - or . or ₁₀	integral of positive number

3) The digits before the exponent may cause the usual overflow reaction of section 10.9.

4) If the absolute value of the exponent is less than 511, but the resulting number is outside of the range of reals, the program execution is terminated with a spill-alarm.

5) If the absolute value of the exponent is greater than 511 the procedure will yield a wrong result or call the spill-alarm, depending on the numeral.

These rules are illustrated by the results of reading the following input string:

34.56₁₀01;-7₁₀+1y+8z.9v₁₀-2w-₁₀2s-.5t
12-3.45+-.+₁₀02+a3₁₀+1..₁₀.a+b.2.3₁₀₁₀c
₁₀-+₁₀-.4de₁₀-₁₀1₁₀-f₁₀-2-₁₀-1m

by means of several calls of read real. The results and corresponding values of char are:

Numeral	char	Numeral	char
34.56₁₀01	133	-3.45	160
-7₁₀+1	24	₁₀02	160
8	25	3₁₀+1	59
.9	21	.2	59
₁₀-2	22	.3	155
-₁₀2	18	-.4	52
-.5	19	-₁₀1	155
12	32	₁₀-2	32
		-₁₀-1	36

10.12. STANDARD PROCEDURE: read general.

10.12.1. Implied procedure heading

```
integer procedure read general(R, SKIP AND EXIT, ELEMENT);
value SKIP AND EXIT; <type> array R;
Boolean SKIP AND EXIT; integer ELEMENT;
```

10.12.2. Example

```
w := read general(I, 3 0 7 27 3 2 7 64 3 1 7 5 3 3, q)
```

10.12.3. Semantics

The purpose of this procedure is to provide fast input of complicated data formats by making it possible for the procedure to read a fairly long strech of characters without the need to make a new procedure call every few characters, at the same time as it assigns all of the information of the input string to the elements of an array in the form of a mixture of numbers and characters.

These needs can only be satisfied by a rather complicated procedure. It is believed, however, that the procedure will be useful for a wide range of needs, from simple input of arrays to input of flexible character strings for general data processing.

The action of the procedure may be described briefly as follows: In a normal call the procedure will read a series of numerals and the characters appearing between them. The numerals will be converted to internal representation and assigned to elements of an array. The intervening characters will be treated as follows: Apart from blind characters every character which immediately follows a numeral, but is not part of it, will act as a terminator of that numeral. Furthermore all such characters, whether they act as terminators or not, will be processed as follows: Two particular characters, specified by the user, will be skipped. Two other characters, also specified by the user, will cause exit from the procedure. All other characters will be recorded in between the numbers in the array.

Exit from the procedure will take place either (1) when one of the exit characters is found, or (2) when the array is full. Information about the exit situation is available at the return from the procedure call.

In terms of the notation of section 10.12.1 the detailed conventions are as follows:

R is the array into which numbers and characters are read. Each number and each character occupies one element of R. The representations are as follows:
Number into real array: as a real number.
Number into integer or boolean array: as an integer number.
Character: Bit pattern having two integer parts:
Bits 10 to 19: Character with case, as the value of char
- 28 - 39: Pointer to the next following element holding a character, or 0 if there are no more.

SKIP AND EXIT is a bit pattern defining four particular characters, denoted as follows:

Position	Character
1 - 9	SKIP 1
11 - 19	SKIP 2
21 - 29	EXIT 1
31 - 39	EXIT 2

The 9 bits describing a character consist of a group of 2 bits and one of 7 bits. The leading 2-bit group of each character description indicates the case to be associated with the character, as follows:

2-bit value	Case of character
0	Lower
1	Upper
2	Lower or upper
3	None existing, i.e. ignore description

The 7-bit group, if interpreted as a binary integer, gives the internal representation of the character, as defined in section 10.3.

The effect of the SKIP AND EXIT parameter is as follows: None of the characters described by the parameter will ever be read into elements of the array R. The two SKIP characters will simply be skipped, while the two EXIT characters will cause exit from the procedure. All of the four characters will, however, act as normal terminators of numerals.

The effect of setting one of the SKIP AND EXIT characters to be one of the characters: <digit>|.|-|+ depends in a complicated manner on the circumstances and should be avoided.

ELEMENT at the call of the procedure must be 0 or point to the last used element of array R. The procedure will start assigning a number or a character to the element at this starting value +1. At exit from the procedure, ELEMENT will point to the last element to which the procedure has assigned. The values of ELEMENT correspond to a simple numbering of the elements of R from 1 and upwards, independent of the dimensions or subscript bounds of the array.

Upon exit the situation is described by the values of char and ELEMENT, and by the value of the function designator:

Value of read general	Situation at exit: Characters are stored	char is an EXIT char.	Array is full
-1	No	No	yes, or ELEMENT is < 0
0	No	Yes	Maybe
>0	Yes, value points to the first	Maybe	Maybe

Note particularly that if the exit is caused by EXIT 1 or EXIT 2, the exit character is found in char and may cause another exit immediately at the first following call.

Further details of the conventions are as follows:

10.12.4. Numbering of array elements.

The elements of R are pointed to in three ways: (1) by values of read general, (2) by bits 28 to 39 of character words, and (3) by values of ELEMENT. These pointers all refer to a numbering of the elements from 1 and up, irrespective of the subscript bounds and dimensions of the array. The number of each element in the general case of an array A[L1:U1, L2:U2, ... Ln:Un] is defined as that number which would be assigned to the element by the following process:

```
number := 1;
for i1 := L1 step 1 until U1 do
for i2 := L2 step 1 until U2 do
  . . . . .
for in := Ln step 1 until Un do
    begin A[i1, i2, ... , in] := number;
    number := number + 1
    end
```

10.12.5. Numeral of type integer

When the type of the array R is integer or Boolean, the numerals recognized in the input are the same as those recognized by the procedure read integer (cf. section 10.10).

10.12.6. Numeral of type real

In the case of real array R, a numeral in the input consists of a signed, decimal number, but without exponent. The numeral must contain at least one digit. The first digit is interpreted as follows:

Characters preceding first digit		Interpretation of first digit
Second	First	
not minus	.	Fractional of positive
-	.	Fractional of negative
anything	-	Integral of negative
anything	not . or -	Integral of positive

Following a fractional first digit the reading continues until the first non-digit is found. Following an integral first digit the reading continues until the first non-digit is found. If this is a decimal point, the reading continues in the fractional mode, otherwise the numeral is terminated. In either mode, the digit causing overflow will be treated as a terminator.

10.12.7. Special treatment of point, plus and minus

Where the characters point, plus and minus do not form a part of a numeral, they act as terminators and are stored in the array R like other characters, with the additional special rule that the two character pairs +. and -. are treated as distinct characters having values derived by adding 256 to the values corresponding to + and -, i.e.

+. corresponds to 416
-. - - 288

10.12.8. Character test sequence

The behaviour of the procedure in singular cases, such as setting a SKIP or an EXIT character to be a blind character, depends on the arrangement of the sequence of tests applied to each character taken from the input. Questions of this sort may be answered on the basis of the following character test sequence.

Test number	Class of character	Action
1	Part of numeral	Number conversion; go to next character;
2	SKIP 1 or 2	Terminator; go to next character;
3	EXIT 1 or 2	Terminator; go to exit action
4	Upper or Lower case	Record case; go to next character
5	Blind	go to next character
6	All others	Terminator; Store character in array; go to next character

The action of the procedure Terminator used in this description must be empty when there is no numeral waiting to be terminated. Again the action of Number conversion includes the termination of a numeral by point, plus, or minus, as described above.

As one useful conclusion which may be drawn from this sequence, it may be mentioned that the normally blind BLANK will be active as SKIP or EXIT character.

10.12.9. Illustrations

Let us choose:

SKIP 1	,
SKIP 2	a and A
EXIT 1	not used
EXIT 2	F

and let us read into

real array Q[22:27]

As the first example, let the program be as follows:

```
    ELEMENT := 0;
rg := read general(Q, 3 0 7 27 3 2 7 49 3 3 7 0 3 1 7 54, ELEMENT)
```

and further let the input string be:

```
    22.0, 23.2, 24.3, 25 a A  26, 27,
```

then the situation after the call will be:

ELEMENT = 6, rg = -1, char = 27, the elements of Q = 22.0, 23.2, 24.3, 25.0, 26.0, 27.0.

Second example:

```
Program: ELEMENT := 3;
rg := read general(Q 3 0 7 27 3 2 7 49 3 3 7 0 3 1 7 54, ELEMENT)
Input string:
    23.,.38AF
```

After call: ELEMENT = 5, rg = 0, char = 182 (=F). The elements of Q are unaltered, except for Q[25] = 23.0 and Q[26] = 0.38.

Third example:
Program as in the first example.
Input string:
2, .7 -. A f -. 3 BD
The situation after exit:

Variable	Value		Comment
ELEMENT	6		Q[27], the last element read into
rg	3		Q[24], the first character
char	180		D
Q[22]	2.0	1	
Q[23]	0.7	2	
Q[24]	288, 4	3	Character -. and ref. to Q[25]
Q[25]	54, 6	4	- f - - - Q[27]
Q[26]	-0.3	5	
Q[27]	178, 0	6	Character B, the last one

10.13. STANDARD PROCEDURE: read string.

10.13.1. Implied procedure heading.

```
integer procedure read string(S, START AND EXIT, ELEMENT);
value START AND EXIT; <type> array S;
Boolean START AND EXIT; integer ELEMENT;
```

10.13.2. Example.

```
if boolean read string(A,10 384 10 17 10 384 10 145,n) then Q
```

10.13.3. Semantics.

The procedure reads a sequence of characters and records them as one string into an array. All characters in the input are skipped until one of two particular characters, specified by the user, is found. Two other characters, also specified by the user, will cause exit from the procedure. Except for one special situation the procedure will start by assuming the input to be in lower case.

The detailed conventions are as follows:
START AND EXIT is a bit pattern defining the four particular characters mentioned above. The four characters, denoted START 1, START 2, EXIT 1 and EXIT 2 are packed in the same manner as in the parameter SKIP AND EXIT in the procedure read general (cf. section 10.12.3). Apart from two situations all the characters in the input are skipped until a character, defined by START 1 or START 2 is found. The two exceptions are:

1. The 2-bit groups of START 1 and START 2, defining the case to be associated with the characters, have both the value 3.
2. The value of the parameter ELEMENT is on entry equal to the number of elements of the array S. The procedure will, set ELEMENT to zero and assume the input to be in opposite case of that associated with the standard variable char.

The following characters in the input are treated as follows:
1. Each character is tested against EXIT 1 and EXIT 2.
2. TAPEFEED and ALL HOLES are ignored.

3. The internal value of CARRET is changed to 63, whereas characters having the value > 64 will be treated modulo 64.

4. The characters will be packed into successive elements of the array S, each element containing 6 characters (cf. section 7.7.2).

5. The characters UPPER CASE and LOWER CASE appear as in the input, except that superfluous case shifts are removed.

Exit from the procedure will take place either when one of the characters EXIT 1 and EXIT 2 is found or when the array is full. If one of the exit characters is found the actions are:

1. If the last case packed is an UPPER CASE, a LOWER CASE is packed as the next character. If this is impossible because the array is full, exit is done with the function designator having the value -1.

2. Bits 0-3 and the remaining 6-bit groups in the current element of the array are filled with characters having the value 10, and exit takes place with the function designator having the value 0.

If an exit character is not found before the array is full the function designator is set to -1. At exit from the procedure, ELEMENT will point to the last element of the array S to which the procedure has assigned. The numbering of array elements is as for the standard procedure read general (cf. section 10.12.4).

10.13.4. Illustration.

The following statements put a text terminated by END CODE on the backing store.

```
place:=n:=char:=0; r:=-1;
for place:=place+1 while place<free size ∧r=-1 do
 begin r:=read string(A,30 384 10 268,n); put(A,free area,place) end;
```

place

11. STORAGE ADMINISTRATION DURING PROGRAM EXECUTION

In the present chapter we shall discuss the handling of the non-homogeneous store of Gier, as this takes place during the execution of Gier Algol programs, and the way this may be controlled by the programmer. The discussion is complicated by the fact that several different Gier configurations exist.

11.1. GIER STORAGE UNITS.

The storage units under consideration are:

1. The core store, or working store, of 1024 words.
2. The drum, or drums, consisting of 1, 2, or 3 drums having each a capacity of 320 tracks, each of 40 words.
3. The buffer store, of 4096 words.
4. The disk file. This may either replace the drum and then has 9600 blocks of 40 words, or it may communicate only with the buffer store, having then 1200 blocks of 400 words.
5. Magnetic tape units, communicating with the buffer store in blocks of up to 4096 words.
6. Caroussel unit, communicating with the buffer store in blocks of 512 words.

The machine always has the core store and a drum or a disk file replacing it. All other storage units are optional.

Information about the machine configuration actually at hand may be obtained by a call of system (cf. section 8.2 and appendix 4).

11.2. STORAGE OF VARIABLES.

Gier Algol 4 includes the option to have the elements of arrays stored either in the core store or in the buffer store, if this is available. Other variables of the Algol program and certain administrative parameters will be stored in the core store. The reservation of core and buffer storage for a variable is made at the time of entry into the block in the head of which the variable is declared. Similarly reservations for a block are cancelled at the time of the corresponding exit from the block. For this reason the space reserved for the variables will usually change from time to time during the execution of a program, being at every moment equal to the sum of the reservations made by those blocks and procedure bodies which are active.

The reservations made at a block entry may be derived from the declarations (including the implicit ones for local labels) of the block, as follows:

	Number of locations reserved
Simple variables, local labels, local switches, local procedures	One in core store for each
Array segment	In core store: number of array identifiers + 1 + number of subscripts. In core or buffer store: total number of variables
Working location	Depends on structure of program, usually only a few in core store
Block, procedure body	In core store: 2 if normal block, 3 if procedure, 4 if type procedure
Formal parameter	In core store: 2 if the corresponding actual is a constant, otherwise 1.

The total number of core store locations available for reservations is about 650, the rest being used for program. For reasons given below it usually is unwise to reserve that many locations in the core store. The full capacity of the buffer store, 4096 words, may be used for array elements, unless part of it is reserved for systems added to the basic Gier Algol 4.

11.3. STORAGE OF PROGRAM.

During program execution, the translated Algol program is permanently stored on the drum in the form of program segments, each of one track. The part of the program being executed is also present in the core store. The transfer of program segments from the drum to the core store is handled by a fully automatic administration. A detailled description of this administration is found in P. Naur: The performance of a System for Automatic Segmentation of Programs Within an Algol Compiler (Gier Algol), Comm. ACM 8 (1965), 671 - 677. In this system an attempt is made to always make the best use of that part of the core store which is not currently reserved for variables. This section of the core store will be divided into program track places, each of 41 locations. The available places will be used for those program tracks which are required as the program execution develops. Whenever the program execution calls for a transfer to another track it is investigated whether the track is available in the core store. If it is not, it is transferred to that track place which for the longest time has been left unused. Consequently the program segments comprising a loop which is short enough to be held in the available core store will only be transferred from the drum when the loop is entered, and not during subsequent repetitions.

The transfer of a drum track to the core store requires 20 milliseconds, and a corresponding transfer from the disk file between 17 and 75 milliseconds. In contrast the transfer of control to a track which is already present in the core store takes between 0.7 and 1.6 milliseconds. It is therefore clear that the program execution time may become highly

dependent on whether the more active loops of the program may be held in their entirely in the core store or not. But this again depends on how much storage is reserved for variables. We can therefore state the two rules for the programmer who wishes to achieve high speed of execution:

1. MAKE THE TEXT OF THE MOST ACTIVE LOOPS SHORT
2. AVOID OVERLOADING THE CORE STORE WITH VARIABLES

As a crude guide, keep the number of reserved locations below 500. This may be achieved by using the backing store for data.

In estimating the length of program loops the segments used in calling standard procedures must be included. To enable this to be done, here is the list of the contents of the tracks of the basic standard procedure library. Where a standard procedure is listed more than once, it uses more than one track.

Standard procedure library track	Procedures
5	read integer, read real
6	read real
7 - 8	read string
9 - 11	read general
12	write integer
13 - 15	write
16	write, writetext
17	cos, sin, sqrt
18	arctan, sign
19	ln, system
20	exp, checksum
21	put, get
22	where
23	where, reserve, cancel
24	reserve, cancel
25	reserve
26	il, us

As a further aid to programmers who wish to make sure that only a small part of the execution time is spent on transfers of segments from the drum, the system includes a standard

integer tracks transferred;

at every drum transfer of a program segment this is increased by unity. The initial value at program entry is zero.

11.4. LOOP STORAGE CONTROL.

To enable the programmer to avoid that short, frequently executed loops are placed across a segment limit by the translator, thereby giving rise to a large number of segment transitions with a resulting increase of execution time, the system allows the programmer to indicate that particular loops should be placed exclusively on one segment. The loops to be treated in this way must be written as for statements with a for clause containing either just one for list element, or two elements of the form

<arithmetic expression> , <arithmetic expression> while
<Boolean expression>

The statement following the for clause must be enclosed in the begin-end pair. To assure storage on only one segment, the programmer must write for as the last symbol of the comment following the end. According to Algol 60 this will be part of the comment, but it will influence the storage in Gier Algol 4.

The use of this facility will sometimes cause that part of a segment is left unused by the translator, and so will increase the total size of the translated program. For this reason it should only be used on such short loops which contribute significantly to the execution time.

Example:

```
for i := 1 step 1 until N do
    begin sum := sum + A[I]↑2 end for;
```

If the symbol for is found following end in any other context, it will give rise to an alarm during translation.

11.5. DATA STORAGE ON BACKING STORE.

Use of the drum, the disk file, or other auxiliary stores, for storage of data, requires that the user includes explicit calls of certain standard procedures. Here are the identifiers and main characteristics of these procedures.

Identifier	Example, reference	Effect
reserve	reserve(‡<jj34‡, 8) section 11.7	integer procedure reserve makes a reservation of an area and enters a corresponding name and description in the catalogue.
where	q:=where(‡<jj34‡, W) section 11.8	integer procedure where searches the catalogue for a given name and yields the description of the corresponding area.
cancel	s:=cancel(‡<jj34‡) section 11.9	integer procedure cancel removes a given name from the catalogue and releases the corresponding area.
put	v:=put(A, W, t) section 11.10	integer procedure put transfers the data held in an array, A, to the tracks of an area W indicated by an integer t.
get	p:=get(A, W, t) section 11.10	integer procedure get transfers the data held in the tracks of an area, W, indicated by t, to an array A.
il	b:=il(B, 130, false) section 11.13	Boolean procedure il calls for an execution of the machine instruction il.
us	us(Q, 7, w) section 11.13	procedure us calls for an execution of the machine instruction us.

11.6. BACKING STORE AND CATALOGUE.

The machines are equipped with a backing store organized as a number of blocks thus:

either (1) a drum which physically may be a disk, block length 40 words, corresponding to one track,

or (2) a disk connected to the buffer. In existing machines the block length is 400 words, but in the future even larger blocks may be possible.

This equipment gives room for an indefinite number of separate, named backing areas:

1) The free area, name: ⟨<free⟩.

2) Reserved, named areas which originally have been part of the free area.

3) Other named backing areas.

These areas are described in the catalog which always is placed on the drum. The catalog may also contain descriptions referring to other kinds of store (magnetic tape, caroussel, etc.), but the procedures put and get are only designed to handle backing areas.

The catalogue and the contents of the named areas may keep data indefinitely for days or weeks unless overflow of storage capacity or machine malfunction breaks in. From within an Algol program the user may communicate freely with any backing area of which he knows the name, he may cancel the reservation of named areas, and he may reserve and name any number of new areas. During a run of one program the named areas stay in the same locations, but between two runs the use of the store may possibly have been reorganized and the catalogue changed accordingly. Between one run and the other the user may therefore only identify an area by its name, but within the same run an area description may be used. In order to communicate with an area containing data stored away at an earlier run, a program must therefore start with a call of a standard procedure, where, which provides the current area description when supplied with a name. In the actual transfers of data between the area and the variables of the program the area description is used. These transfers use the procedures put and get. New, named areas, which are established by means of a call of procedure reserve, are always taken at the beginning of the free area ⟨<free⟩, which is reduced correspondingly. Such reserved areas may be returned to the system by calls of a procedure, cancel. This will not reorganize the store, however, but only cancel the catalogue entry.

A name of an area is a string.

11.7. STANDARD PROCEDURE: reserve.

11.7.1. Implied procedure heading.

```
integer procedure reserve(NAME, BLOCKS); value BLOCKS;
    string NAME; integer BLOCKS;
```

11.7.2. Example

```
action := reserve(⟨<pn23⟩, w)
```

11.7.3. Semantics

Each call of reserve will attempt to reserve an area of BLOCKS blocks and to enter NAME and the description of the area in the catalogue. If succesful, the area thus reserved is the first part of the free area, having so far had the name ⟨<free⟩. At the same time the free area is reduced accordingly. Thus, if a description of ⟨<free⟩ had previously been obtained by a call

where(⟨<free⟩, FREE)

this previous value of FREE will describe an area which contains the newly reserved area as the first part of it. To obtain the correct, new description of the free area, the program must contain a new call of where. On the other hand, several areas having different names may be reserved in direct succession, without any intervening calls of procedure where. Indeed, the program need not make any direct reference to the free area at all.

If the reservation is indeed made, the value of reserve will come out as 0. If the value comes out different from 0, the reservation has not been made, for the reason given in the following table:

Value of reserve	Meaning
0	The reservation has been made
1	NAME is not an allowed name
2	NAME is already in the catalogue
3	The catalogue is full
4	BLOCKS < 0 or too large to be accomodated
5	The catalogue has been destroyed

11.7.4. Variable string as name.

Procedure reserve will call the NAME in the way that procedure writetext calls its parameter, so the facilities of section 9.4.4 may be used.

11.8. STANDARD PROCEDURE: where.

11.8.1. Implied procedure heading.

```
integer procedure where(NAME, AREA); string NAME; integer AREA;
```

11.8.2. Example

```
check := where(⟨<pn23⟩, pn23)
```

11.8.3. Semantics

Each call of procedure where causes a search of the catalogue for an item having the given NAME. If the item is found, the corresponding description of its current storage is assigned to AREA. If the item refers to a backing area the value assigned to AREA is suitable to be used in calls of procedures put and get. As a special case, the call

where(⟨<free⟩, W)

places the description of the free area in W.

The value delivered as result of the call indicates the success of the search, as follows:

Value of where	Meaning
0	A backing area description has been assigned to AREA
1	NAME is not found in catalogue, AREA is unchanged
2	Another description has been assigned to AREA
3	The catalogue has been destroyed, AREA is unchanged

11.8.4. Variable string as name.
The rule of 11.7.4 also holds for where.

11.9. STANDARD PROCEDURE: cancel.

11.9.1. Implied procedure heading
```
integer procedure cancel(NAME); string NAME;
```

11.9.2. Example
```
q := cancel(<<pn55>)
```

11.9.3. Semantics
Each call of cancel causes an attempt to find NAME in the catalogue and to cancel the corresponding reservation of an area. In this process the free area is extended to include any unused area which is made to lie adjecent to the current free area.
The success of the call is indicated by the resulting value:

Value of cancel	Meaning
0	NAME is found and cancelled
1	NAME was not found in the catalogue
2	NAME indicates an item in the catalogue which must not be cancelled.
3	The catalogue is destroyed.

11.9.4. Variable string as name.
The rule of 11.7.4 also holds for cancel.

11.10. STANDARD PROCEDURES: put AND get.

11.10.1. Implied procedure headings
```
integer procedure put(A, AREA, PLACE); value AREA, PLACE;
    integer AREA, PLACE; <type> array A;
integer procedure get(A, AREA, PLACE); value AREA, PLACE;
    integer AREA, PLACE; <type> array A;
```

11.10.2. Examples
```
y := put(B, POPULATION, 8)
z := get(age, POPULATION, 32)
```

11.10.3. Semantics

These procedures perform transfers of data between the variables of the program, indicated by means of the array identifier A, and the part of AREA indicated by PLACE. Procedure put transfers from the array to the area leaving the array unchanged, procedure get in the opposite direction, changing the array, but otherwise the conventions are similar. The precise effect of put and get depends on the block length of the backing store.

The value of AREA must have been obtained by a call of procedure where. PLACE must be a positive integer, indicating the first block which is involved in the transfer, the blocks within the area being numbered 1, 2, 3, ... Thus we must have

1 < PLACE < number of blocks in area

and the transfers will generally involve all the elements of the array A and the blocks within the area having numbers PLACE, PLACE + 1, ... Other details are as follows:

1) The elements of arrays of dimension 2 or higher are treated as a linear sequence, as explained in section 10.12.4.

2) Irrespective of the block length, the array A must have at least 40 elements.

3) As many complete blocks in the area are used as are necessary to hold the full array, i.e.

number of blocks =

(number of elements + block length - 1) : block length

The data transferred to the backing store from an array of a given size by a call of put, can be got back by a call of get with the same AREA and PLACE and an array of the same size.

4) The exact arrangement of the elements on the blocks is generally unspecified. However, if the number of elements of the array is an integral multiple of the block length, the elements are mapped directly on the blocks. In this case it is possible to transfer the same data alternatively in terms of arrays of different sizes.

11.10.4. Alarm conditions

The following error conditions will terminate the program execution with an alarm:

Procedure put is called with AREA not describing a backing area.

The array is too large to be accomodated in the area, starting at PLACE.

The array has less than 40 elements.

11.10.5. Hardware failure

In 400-word block machines the hardware gives indications of certain failures of transfers in the form of a so-called status word. These are handled as follows: Following the transfer of each track, the procedures check for a failure indication. If a failure is indicated the procedures exit immediately, even if only part of the track transfers have been made. In any case the result of the procedures is the last status word, or in case of 40-word block machines, zero. If a hardware failure has occurred the status word is negative, otherwise positive.

11.11. ADVICE ON SEMI-PERMANENT DATA STORAGE.

In programming the use of semi-permanent data storage on the backing store it should always be kept in mind that although some attempt has been made to prevent that simple mistakes will cause data to be destroyed unintentionally, the protection of data is far from complete. Programs making use of these facilities should therefore include safety measures, such as the following:

1) Always keep a reasonably simple way open to restore the data left in the backing store. Often these consist of an initial set of data, which has subsequently been changed in a number of updating processes. In this situation it is vital that the initial set and all the subsequent changes are kept in such a form that the complete sequence of modifications can readily be repeated. Also it is advisable to produce copies of the data in its modified form on an external medium from time to time.

2) Start the program right away with calls of procedure where to all the areas used within the program, to make sure immediately that they are all present. Also make some independent check that the contents of each single track is correct. As the minimum, make a summation check of one word of each track.

3) To avoid the double use of names of areas, make sure to follow the conventions adopted for this purpose at each installation. A possible form of such conventions is to compose the name of the initials of the user, two or three letters, and a whole number between 100 and 999, each particular user starting generally at 100 and working up: 101, 102, .. with the possibility to start at 500, say, if two users have the same initials.

11.12. ADVANCED USER INFORMATION.

The conventions of the procedures put and get imply a certain amount of checking against mistakes and are all that are needed by the normal user. The following additional information is given for the benefit of advanced users who are able to avoid the possible ill consequences of bypassing these checks and who need more flexible communication with backing stores.

The value of AREA, as supplied by a call of procedure where, is exactly the first word of the catalog item. In case of a backing area the format is:

Bits 0- 2 0 in case of drum, 1 in case of disc
Bits 3- 7 Specifies the further use of the area, see below.
Bits 8-23 Number of blocks in area.
Bits 24-39 First block of area.

The value of AREA is tested as follows: get and put: alarm if not drum or disc; put: alarm if bit 3 = 1 or bit 5 = 0; cancel: yields the value 2 if bit 3 = 1 or bit 5 = 0; where: yields the value 2 if bits 0-2 ≠ actual backing store; reserve sets the following bit pattern in bits 3-7: 00100 and backing store value in bits 0-2 and inserts a dummy sum word in the catalog item.

For detailed formats of the catalog see the description of the Cat system.

11.13. STANDARD PROCEDURES: il AND us.

11.13.1. Implied procedure headings.

```
Boolean procedure il(A, FUNCTION, PARAMETER); value FUNCTION, PARAMETER;
    <type> array A; integer FUNCTION; Boolean PARAMETER;
procedure us(A, FUNCTION, PARAMETER); value FUNCTION, PARAMETER;
    <type> array A; integer FUNCTION; Boolean PARAMETER;
```

11.13.2. Examples

```
il(C, 130, 20 120 20 1)
us(Q, 7, ((Boolean (spool × 16 + block) shift 10) ∨ 40 2) shift 20)
```

11.13.3. Semantics.

Each call of il or us causes execution of the machine instruction having the same operation code as the identifier of the procedure. These instructions call transfers of data between the buffer store of the machine and any attached magnetic tape drivers, caroussel units, or disk file units. To understand the action of these procedures the user must refer to the description of these units and their attachment to Gier, given in A Manual of Gier Programming Vol. III and later reports.

The action of an il or us instruction depends partly on the effective address of the instruction, partly on the parameter placed in the R-register. The effective address must be given as the value of FUNCTION. The parameter placed in R before the activation of the il or us instruction is formed by adding the buffer address of the last word preceding the array A in the rightmost position of the word given as PARAMETER. This means that bits 28 - 39 of PARAMETER indicates the first element taking part in the transfer by its element number, numbering the elements within the array as 1, 2, ... as in section 10.12.4.

Before execution of the il or us instruction certain parts of PARAMETER and FUNCTION are checked to be mutually compatible and to involve only the elements of A. This check depends on the last 4 bits of FUNCTION, as follows:

1) Magnetic tape operation, i.e. the last four bits of FUNCTION represent a number between 1 and 6. The block size should be given in positions 8 - 19 of PARAMETER. It is checked that this block size does not extend beyond the array A, that bits 0 - 7 and 20 - 27 of PARAMETER are zero, and that FUNCTION is positive and less than 512.

2) Caroussel operation, i.e. the last four bits of FUNCTION represent 7. PARAMETER must supply:

bits	0 - 5	Spool number
-	6 - 9	Block number
-	10 - 15	Zeroes (checked)
-	16 - 19	Number of blocks of 512 words
-	20 - 27	Zeroes (checked)
-	28 - 39	First element taking part in transfer

It is checked that the number of blocks can be held in the array, starting at the first element, and further that FUNCTION is positive and less than 512 and has zero in binary positions 32, 64, and 128.

3) Disk file operation, i.e. the last four bits of FUNCTION are between 8 and 15. PARAMETER must supply:

bits	0	Zero
-	1 - 9	Block size, checked to be at most 400
-	10 - 21	Track number
-	22 - 27	Zeroes (checked)
-	28 - 39	First element taking part in transfer

It is checked that the block can be held in the array, starting at the first element, and that FUNCTION is positive, is less than 512, and has zero in binary positions 32, 64, and 128.

Failure of any of these checks causes an alarm termination of the program execution.

12. MACHINE CODE IN GIER ALGOL 4.

12.1. OVERALL POSSIBILITIES.

There are three ways to include machine code in a Gier Algol 4 program. Machine code may be written as a statement, starting with the symbol code. Such code will be executed from the place among the Algol statements where it is written. Because of the automatic segmentation of programs it must not contain more than 39 words.

Secondly, machine code may be written as the body of a procedure, starting with the symbol code. This admits the special possibility to omit the specification of formals. In the call of the procedure the corresponding actual parameter of such an unspecified formal may be anything. Again only 39 words are admitted.

As the third possibility, machine code may be written as a new kind of declaration, starting with the symbols core code. Whenever the program enters the block in the head of which this code is written, the code is copied from its locations on the segments to locations in the stack in the core store. In order to execute it, the user must call the standard procedure gier(p), where p must be a simple boolean variable, given as the first parameter of the code. Once the code has been transferred to the stack it may be accessed rapidly any number of times. The translator will handle core code pieces of up to 119 words. At run time the core code pieces will require locations in the core store, like simple variables. If there is insufficient capacity, the normal store overflow reaction will take place.

12.2. SYNTAX.

<code statement> ::= <code head><code specifications>
<machine code> e
<code head> ::= code <identifier list>;
<code declaration> ::= <core code head><code specifications>
<machine code> e
<core code head> ::= core code <identifier list>;

where the detailled format of code specifications and machine code are explained below.

The syntax may be illustrated by the following example, which in the right hand column is provided with an indication of the structure.

```
....
     code A, B;                    Code head, with list of parameters

     2, 44                ;        Code
     1, 46, 44            ;        specifications

     is (b2), arn s a2
     ck  0  , nk r e1
e1:  srn    Dt  -1        ;        Machine code
     ck  10 , gr  a1
     e
```

With some restrictions, described in section 12.8, the code specifications and machine code conform to the syntax of Slip. In particular, the comment conventions of Slip apply.

The code specifications supply information about the storage, kind, and type of the parameters. They only serve as a check of the characteristics of the parameters given in the code head, while there is no check that the parameters described in the code head and specifications are used correctly in the machine code. The check of the agreement of head and specifications is intended as a help to the exchange of machine code pieces. In publication such pieces will only include specifications and machine code. In a particular application the user must add the code head. The translator will check that this is consistent with the code specifications. These make more detailled distinctions of parameters than does a normal Algol text and the user of the code may thereby place strict limits on what constitues a correct parameter in each parameter position. This limits the generality of the machine code piece, but it allows the code to be written in a more efficient manner.

12.3. STORAGE ALLOCATION AND ADDRESSING OF ALGOL QUANTITIES.

The storage allocation of quantities and programs within Gier Algol is dynamic, i.e. the final address where an instruction or variable is stored is not determined until the actual execution of the program. Moreover, even during one execution of a program the address of a quantity will not necessarily remain the same from one phase of the program to the other. For this reason addressing of quantities within the machine code itself, such as jumps and references to working variables, should always employ r-relative addresses.

From within the machine code it is possible to refer to those quantities of the surrounding Algol program which are listed as parameters in the code head. In the following we describe first the methods of address calculation and second the meaning of the words accessed at run time.

12.3.1. s-relative addressing.

In general the final machine address of a quantity of the Algol program is found by an algorithm which depends on the block in which the quantity is declared or introduced as formal. The result of the translation is that each quantity is described by two numbers, the DISPLAY REFERENCE indicating the block and the BLOCK RELATIVE address. The final, absolute address is calculated at run time by the following algorithm:

Absolute address:= STORE[DISPLAY REFERENCE] + BLOCK RELATIVE.

In machine instructions this may conveniently be realized, e.g. as follows:

```
is  (DISPLAY REFERENCE)
op  s+BLOCK RELATIVE
```

where op is some operation code. This general addressing method will be referred to as s-relative addressing. Note particularly that this is perfectly general and may be used in all cases, while the use of p-relative and absolute addressing, described below, depend on the particular program structure. A piece of machine program planned to be used generally should therefore employ s-relative addressing for all quantities.

12.3.2. p-relative addressing

In order to make programs shorter and faster, the value of STORE [DISPLAY REFERENCE] corresponding to the currently local block is kept in the p-register. Consequently it is possible to address quantities declared in that block by p-relative addressing:

op p+BLOCK RELATIVE

12.3.3. Absolute addressing

During one execution of a program the addresses of quantities in the outermost block are fixed, and may be calculated during translation.

12.4. SLIP NAMES.

Within the machine code the references to parameters must be written as Slip names. Each parameter may have an a-name, a b-name, and a d-name associated with it, the set (a1, b1, d1) belonging to the first parameter, the set (a2, b2, d2) to the second parameter, etc. The values of any relevant names are supplied automatically by the translator. In case of s-relative addressing of an array identifier the meaning of the names is as follows:

a-name BLOCK RELATIVE address of identifier word
b-name DISPLAY REFERENCE
d-name BLOCK RELATIVE address of dope vector

More explanations of the meaning of the Slip names are given below.

12.5. CODE SPECIFICATIONS.

The code specifications must have one line for each parameter in the order in which these are given in the code head. The specification of a parameter must have the following structure:

<code parameter specification> ::= <addressing> , <kind and type>|
<code parameter specification> , <kind and type>
<addressing> ::= 1|2|3|4
<kind and type> ::= <unsigned integer>

In addition, the code parameter specification may include comments according to the usual Slip conventions.

The meaning of the four possible values of the addressing indication and of the associated Slip names are as follows:

Addressing	Meaning	a	b	d
1	s-relative	Rel.addr.	Display ref.	Rel.array address
2	Absolute	Abs.addr.	Undefined	Absolute array addr.
3	p-relative	Rel.addr.	Display ref.	Rel. array address
4	Stand.proc.	Track no.	Track rel.addr.	Undefined

The addressing indication has two effects: (1) The translator checks that the parameter given in the code heading is indeed declared in the block level implied by the addressing (addressing 4, standard procedure, may be thought of as corresponding to quantities declared in a block outside the complete program). (2) The values assigned to the Slip names are selected according to the addressing, as shown in the table.

The indications of kind and type given in a specification serve exclusively as a check of the parameter given in the code head. A specification may have any number of kind and type indications, but must be written on one line only.

As described below, each kind and type of Algol quantity belongs to a certain class, with a corresponding number. The translator checks that the number corresponding to each code parameter is given as one of the kind-and-type numbers of the corresponding specification.

12.6. CLASSES AND STRUCTURES OF QUANTITIES.

In the following is given, for each class of quantity distinguished in code specifications: (1) The name of the class, (2) Class numbers to be used in code specifications, (3) A definition of the class, where it does not coincide with one recognized in Algol 60, and (4) the meaning of the word addressed through the associated a, b, and d Slip addresses. The following notation is used:

sr

Stack reference, i.e. the current base address of a section of the stack.

c17

The location in the running system which holds the universal value, UV.

c30

The location in the running system which holds the universal address, UA.

absaddr

The absolute address of a quantity.

drumpoint

Bits 10-19: track relative address
30-39: track number

The track numbers run from some starting number up to 1023. The relative address goes from 0 to 39.

General formal, 12. A formal for which no specification is given. This is acceptable when the procedure body is a code statement. The a, b address points to a word describing the actual parameter, as follows:

		Further use, kind and type number
Constants		
integer	pan c30 [UA] t <abs addr>	60
real	pan c30 [UA] t <abs addr> f	61
Boolean	pan c30 [UA] t <abs addr> Z	62
string	pan c30 [UA] t <abs addr> Z f	63
Simple variables		
integer	pa c30 [UA] t <abs addr>	60
real	pa c30 [UA] t <abs addr> f	61
Boolean	pa c30 [UA] t <abs addr> Z	62
string	pan c30 [UA] t <abs addr> Z f	63
label	pa c30 [UA] t <abs addr> Z f	28
Subscripted variable		
integer	ps <sr> , <drum point>	60
real	ps <sr> , <drum point> f	61
Boolean	qq <sr> , <drum point>	62
Other expression, inclusive type procedure without parameters		
integer	psn <sr> , <drum point>	36, 60
real	psn <sr> , <drum point> f	37, 61
Boolean	qqn <sr> , <drum point>	38, 62
string	qqn <sr> , <drum point> f	63
label	qq <sr> , <drum point> f	28
Array identifier		
qq <address of array word - 1>.9 + <dope address - address of array word - 2>.19 + <number of subscripts + 1>.39		64, 65, 66
For further explanation, see direct array, below.		
Switch and procedure identifier, other than expression		
integer	zq <sr> , <drum point>	40
real	zq <sr> , <drum point> f	41
Boolean	zqn <sr> , <drum point>	42
no type	zqn <sr> , <drum point> f	39, 43
switch	arn <sr> , <drum point>	24, 32

label, 20. The a,b address points to a word as follows:

qq(f)<sr>,<drum point>

where sr points to the block in which the label is local. The f mark indicates that the point is a right half word.

switch, 24. The a,b address points a word of the format given above for a general formal of type switch.

Formal label, 28. The a,b address points to a word of the format given above for a general formal, simple variable, or other expression.

Formal switch, 32. As switch, given above.

Type procedure without parameters, integer 36, real 37, Boolean 38. As general formal, other expression.

Procedure without type and parameters, 39. As general formal, switch and procedure identifier, other than expression, no type.

Procedure with parameters, integer 40, real 41, Boolean 42, no type 43. As general formal, switch and procedure identifier, other than expression.

Direct variable, integer 44, real 45, Boolean 46. A simple variable or a formal variable called by value. The a,b address gives the location holding the value.

Direct array, integer 48, real 49, Boolean 50. A non-formal array identifier, or a formal array identifier which has appeared somewhere in the Algol text followed by a subscript list. The a,b address points to the so-called array word:
Array word:
Bits 0 to 39 depend on the translator version:
a) Arrays in buffer store:
qq c17.9 + 1.19 + (address of element 0,0, ... ,0).39
b) Arrays in core store:
qq (address of element 0,0, ... ,0).39
The address of element 0,0 ... , 0 will not necessarily lie within the range of possible machine addresses, since these subscripts may not lie within the subscript bounds.
The marks indicate the type: 0 = integer, b = real, a = Boolean.

The d,b address points to the dope vector, which consists of several words. If the array declaration is: array A[l1:u1, l2:u2, ... , lp:up], and we define ci = ui - li + 1, then the dope vector consists of:
dope address - 1: constant term = (((..(l1×c2+l2)×c3+l3) ...)×cp+lp
in position 39
dope address: length = c1×c2×c3 ... ×cp in position 39
dope address + 1: c2 in position 39
....
dope address+p-1: cp in position 39

Name variable, integer 60, real 61, Boolean 62. A formal simple variable, called by name. Depending on the corresponding actual parameter, the a,b address will point to one of the words given for general formals, constants, simple variables, subscripted variables, or other expression.

Formal string, 63. Depending on the corresponding actual parameter, as given for general formal, constant, simple variable, or other expression.

Indirect array, integer 64, real 65, Boolean 66. A formal array identifier which has nowhere appeared followed by a subscript list. The a,b address points to a word as given for general formal, array identifier.

Standard procedure 68. The a, b names correspond to addressing 4, section 12.5. They point to the entry point of the standard procedure. The track number is relative to a base address which is not directly available to the user.

12.7. CORE CODE AND STANDARD PROCEDURE gier.

The first parameter of any piece of core code must have the description:
3, 46
indicating p-relative addressing and direct Boolean variable. When at block entry the core code piece is transferred to the working store, a description of the entry into the core code is also assigned to this variable. In order to activate the core code, standard procedure gier should be called with this variable as parameter. The parameter of procedure gier should only be variables which have occurred in this special way. Standard procedure gier has the following implied procedure heading:

integer procedure gier(u); Boolean u;

Activations of core code from within blocks or procedures called at the level of the code itself should only be done with great care. It must constantly be kept in mind that any activation of a block or procedure will change the current values of the p-register and the display and therefore will influence the proper addressing of Algol quantities.

12.8. MACHINE CODE FORMAT.

The machine code which may be written within the Algol program is a proper subset of Slip, i.e. it conforms to Slip conventions, but does not provide all of the facilities of Slip. In the following list we give the constructions which are not admitted.

Slip facilities not admitted in machine code:

Construction	Explanation
b	No inner blocks
c	Only limited redefinition of i, see below
g	No dummy information
h	No call of HELP
l	No change of medium
n	Unnecessary since r is not admitted
r	No automatic relative addressing
u	No direct exit from the code
x	No label table administration
z	- - - -
<	No transition to binary code
h in label	No special marking of right half instructions
₁₀	No exponent
/	No integer groups, no / between instructions
<	No conditions
m in address	No automatic relative addressing
k	No references to track numbers
terminator	Numbers must be terminated by one a, b, c, or <empty>
number-line	Only one number per line
definition line	Only one definition per line
i	The serial address can only be increased

Violation of these rules will cause an alarm during translation.

With respect to labels the machine code must be written such that it would be correct Slip if it had been preceded by the following two block heads:

```
b c127
<definition of c-names, see below>
b a127, b127, d127, e127
```

There is, however, no check that names beyond these are not used.

Certain of the c-names are defined when the machine code is processed and give access to the Running System, i.e. the run-time Algol program administration. For a full understanding of the possible uses of these names, a knowledge of the details of this administration is necessary. Here we give only the names and a corresponding key word.

c0 display 0
c1 next track
c2 go to track
c3 next param track
c4 call std. proc
c6 call rel track
c7 prepare block or call
c8 expression as formal
c9 exit block
c13 go to computed
c17 UV = universal value
c18 end UV,R,RF expression
c19 end addr expression
c20 reserve array
c24 assign to formal subsc.
c26 go to point in R
c27 error

c30 UA = universal address
c33 address, working location
c35 last used
c37 next in
c39 next out
c42 qq 1.39
c44 qq -1 t 256, 0.5 floating
c49 base track table
c53 qq 0 t 256 = 1.0 floating
c54 char, section 10.7
c55 vy last select t mask, section 8.1
c57 qq appetite t -1
c58 qq 10.39
c61 tracks transferred
c63 get place
c64 catalog
c65 current place

The value of a c-name defined in one piece of code survives to all the following pieces of code within the same program.

Return to the execution of Algol statements must be done by means of the following, or an equivalent, instruction:

hv value of s at entry + 1.

When the return is made, the register p must have the same contents as at entry, while all other machine registers may have been changed arbitrarily. In case of core code, the value of the execution, i.e. the value of the call of gier, must have been placed in the R-register.

13. COUPLING TO ENVIRONMENT.

13.1. GIER ALGOL SYSTEMS.

In order to be able to accomodate the translator and running system in any of a number of machine configurations, the Gier Algol 4 system is written as several programs which may be combined and used in several ways. To be directly useful it is necessary that these are handled by a certain, although modest, amount of additional programming. It is anticipated that different installations will have different needs, and will develop their own Gier Algol systems accordingly. The detailled user conventions of these systems will have to be obtained from separate documentation. The present section only describes the basic conventions, to be used in developing such user oriented systems. Any system will have to distinguish between translation and execution of Algol programs. These two problems are discussed separately below.

The descriptions refer to the following kinds of store:
Drum, which physically may be a disk.
Buffer medium, i.e. a storage accessed via the buffer. Physically it may be a disk, a carroussel, or a magnetic tape.
Paper tape
Type writer input

13.2. TRANSLATION.

The translation consists of a combination of several collections of data and storage areas, as described in the following sections.

13.2.1. Algol program text.

The Algol program text must be supplied from paper tape, from typewriter input, from the drum, from a buffer medium, or from a combination of these. In the case of paper tape or typewriter the characters are supplied directly one by one. In the case of the drum or a buffer medium the characters must be packed into the words in the manner described in sections 7.7.2 and 9.4.4. The text must start in the first word of a block and must continue in the following words of that block and on following blocks in an obvious manner.

The starting medium must be given as one of the translator initialization parameters of section 13.2.5. The change from one medium to another requires the catalogue system Cat to be present in the machine and the relevant media to be present in the catalogue. The selection of another medium is made whenever the text currently being translated contains a copy call:

<copy call> ::= copy <copy source><

with

<copy source> ::= <any sequence of characters not containing a < character, i.e. character value 17>

A copy call may be written between any two symbols of the Algol program. It will cause the continued input to be taken from the medium or area described in the catalogue item having the name derived from copy source by removing all BLANKs.

The names of the tape reader and typewriter are:

	Catalogue name
Tape reader	r
Typewriter	t

If a piece of program text is to be used as an insertion into another text by copying during translation it must terminate with the symbol finis
This will cause a return to the text from which the piece was called by copy, at the character immediately following the copy call.

A text called by a copy call may call further copying, up to a limit of six levels. If more levels are called, the translator will call an alarm.

13.2.2. Gier Algol 4 translator.

During translation the translator including a standard procedure library must be available on the translator medium which may be the drum, a buffer medium, or paper tape. Including the normal library of about 1000 word the translator will occupy about 6800 words.

A translator stored on paper tape will be taken into the machine in short sections which will be done with successively. This is known as the transient mode of operation.

13.2.3. Working area.

An area of working storage must be available on the drum. This area will also receive the translated program as described in section 13.2.7.

13.2.4. Translation-finished action.

Machine instructions defining the action to be taken when the translation is finished must be stored on a track of the drum, starting at a left instruction at a given relative address of the track. For the situation at entry to this action, see section 13.2.7.

13.2.5. Translation parameters.

When the translator is called, the core store must contain a set of parameters defining the storage of the above data collections and certain variants of the translation process. These parameters must be stored in consecutive machine locations, starting in address 3+e4, where e4 is a value associated with the translator and usually set to 15. The location format and meaning of the parameters are given below. In this description the concept of track number is generally defined as

960 × group number + track number within group.

In configurations without a disk file the group number is 0.

3e4: qq line interval.39

As an aid during detection of programming errors, the translator may be instructed to produce an output of every line interval-th line of the program text.

This parameter is only relevant when the boolean lineprint wanted is set to true; see parameter 10e4.

4e4: qq number of tracks in working area.39; see section 13.2.3.

5e4: qq first track of working area.39; see section 13.2.3.

6e4: qq catalogue look-up track.9 + init medium track.19;
The catalog look-up program is needed whenever a <copy call> occurs in the Algol program text. The program for init medium is needed whenever a buffer medium is used as input medium.
When present these programs must be stored in track group 0. The absence of one or both programs is indicated by the corresponding tracknumber being zero.

7e4: qq initial input medium;
The contents of 7e4 must be given as follows:

Typewriter or paper tape reader: qq -1.2 + by-value suitable for medium.19

Area on drum or buffer medium: The format is as the first word of a text area (see description of the Cat system) with number of blocks replaced by number of characters to be skipped. A simple area on drum may thus be given by:

qq <no of skipped characters>.23 + <first track>.39

Provided no direct references to the catalogue are made, the translator may be called even when there is no catalogue present.

8e4: qq translation finished track.39 + relative address.9
See section 13.2.4.

9e4: qq normal out.6 + error out.16 + type out.26 +
type in.29 + alarm out.36 + secondary reader.39
This parameter must supply a set of values of the by-register (medium selection) for control of input and output media during translation. The uses of each of these media are given below.

Medium	Uses of medium
normal out	Prelude, i.e. copying of the program text, up to the first begin
	line output, see 3e4 above
	Pass information, see section 13.3
	Pass output, see section 13.4.
error out	Messages about program errors which do not immediately terminate the translation
type out	Messages to the operator, requiring action immediately
type in	Operator action
alarm out	Messages about program conditions which terminate the translation immediately, and message output, see section 7.1.7
secondary reader	Transient input of translator

10e4: qq skip input between PUNCH OFF and PUNCH ON (see section 7.1.6).2+
pass information wanted (see section 13.3).4 +
line print wanted.5 +
disk mode.6 +
execution time check of subscript bounds wanted.7
Each of the five Boolean parameters to be placed in positions 2, 4, 5, 6, and 7, must be represented as 1 for Yes and 0 for No.

Diskmode refers to a translator mode which may reduce the number of head movements during translation of large programs on a drum disk. When diskmode is true the requirements to the size of the working area will be greater than for discmode false.

11e4 - 14e4 Parameters which describe the translator medium as follows.
Drum:
11e4: qq first track.39
12e4 - 14e4: irrelevant.
Paper tape:
11e4 - 14e4: irrelevant
Buffer medium: 4 words describing how to transport a block from the medium to the buffer.
11e4: Increment to current block parameter to get next block.
12e4: Current block parameter; i.e. the parameter used in R during the most recent transport of a block.
13e4: qq block length in words.39
14e4: qq il check, unit;
The il-addresses used for statusword transport and block transport respectively.

13.2.6. Call of translator.

With the preparations described in the preceding sections completed, the translator is called to action by transferring its first n words from the translator medium to locations 184e4 and following of the core store and transferring the control to its first instruction.

The value of n depends on the translator medium and is given in bits 0 to 9 of the first word of the translator; it is about 400-440 words.

The binary sum of the n words, including marks added as mark A.8 + mark B.9 is zero.

If the translator medium is paper tape the translator comes in two parts (section 14.2). The first of these must be read before, the second after the Algol program text.

During its work, the translator will make free use of the core store, of the working area, and if a buffer medium is involved, of the buffer store.

The translation is done in 9 separate passes. In each pass the text produced by the previous pass is taken from the working store and a transformed version of it sent back to the working store. In addition certain other actions are taken at various stages. Depending on the Algol program text the translator will issue messages, as described in appendix 3. Briefly, the translation proceeds as follows.

Pass 1. The Algol program is converted to a string of symbols, roughly corresponding to the basic symbols of Algol 60, and transferred to the working area.

Pass 2. Identifiers are replaced by internal symbols. A table of identifiers is built up in the core store, while the program is transferred back to the working area.

Pass 3. Phase 3.1. With the identifier table still in the core store the standard identifiers are taken from the first part of the library. A list of the internal symbols for the standard identifiers is added at the end of the output from pass 2.

Phase 3.2. The program is checked for short span syntactic errors and converted to a form more suited for the subsequent processing.

Pass 4. In a backward pass the declarations of identifiers and certain details of their use in the program are collected in a list at the begin of each block.

Pass 5. Phase 5.1. The description of the kind, type, and addressing of each identifier is distributed to each place in the program where the identifier is used.

Phase 5.2. The descriptions of those standard identifiers which are actually used in the program are taken from the second section of the library and collected in a table in the core store, for use by pass 6.

Pass 6. The expressions and statements are checked for the consistent use of operands and operators. At the same time expressions are converted to reverse Polish form.

Pass 9. Machine code in the source text is converted to internal form. If the program does not include machine code this pass is omitted.

Pass 7. Expressions are converted to final machine instruction form.

Pass 8. Phase 8.1. The program text is rearranged in the working store.

Phase 8.2. The standard procedures used in the program are taken from the library.

Phase 8.3. The program is arranged in segments on the tracks and provided with internal references.

Phase 8.4. The running system segments are added to the program.

The pass numbers are used to identify the source of messages output during translation. In fact, preceding the first message of any kind issued from any pass the pass number followed by point will be output.

In addition to the parameters given in section 13.2.5 the current values of the manually operated KA and KB registers of the machine influence the translation, as follows.

KA: stop before each translation pass or phase.

KB: produce pass output, as described in section 13.4.

The values of KA and KB may be changed arbitrarily during translation.

13.2.7. Translation completed.

At the completion of its task the translator will transfer the translation-finished track to location 224e4 and transfer the control to the location in it indicated as a relative address in the translation parameter 8e4. If the translator has detected illegal conditions or program

errors, the contents of R will be 0. In this case the translation is not continued beyond pass 9. Otherwise R will contain the location of the resulting translated program in the form:

qq number of tracks.23 + first track.39

These tracks will lie within the working area specified for the translation, i.e. they will always consist of tracks of 40 words.

13.3. PASS INFORMATION.

Depending on the setting of the translation parameter in 10e4, an output of information about the program being translated is produced on the normal output medium specified in 9e4. The output which may thus be selected consists of a few integers for each pass, printed in one line when the pass is completed. The first integer, A, in all passes gives the number of tracks produced as output by the pass. The remaining integers have the following meaning (if a parameter is 0 nothing is printed):

Pass 1. The figures refer to the storage of long strings in the working area:
- B. 1024 - the number of tracks or part of tracks used.
- C. The number of words used on the last string track.

Pass 2.
- B. The number of different identifiers in the program.
- C. The number of words used for storing long identifiers.

Pass 3.
- B. The number of blocks in the program.

Pass 4.
- B. The maximum depth in the stack used for collecting the declarations.
- C. The maximum level of nesting of blocks.

Pass 5.
- B. The number of occurrences of identifiers in statements and expressions integer divided by 10.
- C. The number of redeclarations of identifiers.

Pass 6.
- B. The maximum number words used in the operator stack, or 5 if that number is less than 5.
- C. The maximum number of words used in the operand stack.

Pass 9.
- B. The maximum code size.

Pass 7.
- B. The maximum number of words used in the operand stack.

Pass 8.
- A. The number of tracks representing the active program.
- B. The total number of tracks, including A, running system, standard procedures and strings.
- C. The group number of the first track of the final program
- D. The track - - - - - - - - -

13.4. PASS OUTPUT.

While KB is on the translator will produce the so-called pass output on the normal output medium specified in the parameter in 9e4 (see section 13.2.5). The pass output is the output produced internally by each pass, representing a modified form of the Algol program. The print out may be used as the last resort in pinning down troubles in using the compiler, whether these are due to programming errors or faulty machine operation. The interpretation of the pass output requires some insight in the internal working of the translator and will be given in separate documentation.

13.5. EXECUTION.

The text of the translated program, located at the end of translation as described in section 13.2.7, is self contained and may freely be moved about in the machine as long as it is kept as one complete whole. For execution the text must be available on a series of consecutive 40-word tracks of a drum or a disk file replacing the drum. The execution requires the following preparations:

13.5.1. Execution message medium.

The medium to be used for output of messages from the running system must be placed in the by-register.

13.5.2. Execution end track.

A track of instructions defining the action to be taken when the execution is completed must be present on a 40-word track. The situation existing when this action is called is given in section 13.5.5.

13.5.3. Execution parameters.

The following parameters must be set in the core store, locations 259e4 to 264e4.

259e4: qq first buffer location.39 ;
Together with the next parameter, this defines the part of the buffer store, if any, which may be used for array elements (see section 11.2).
260e4: qq first buffer location beyond the available part.39;
If the translator version used during translation places arrays in the core store this parameter must be 0. Otherwise it must be unity larger than the last buffer address available for the program.
261e4: qq tracks occupied in top of the free area.39;
262e4: qq catalogue look-up track;
The catalogue look-up program must be stored in track group 0, if it is present. Otherwise this parameter must be 0.
263e4: qq execution end track (section 13.5.2).39;
264e4: qq first track of translated program.39;

If the translated program is placed in the top end of the free area this parameter should be the number of tracks in the program. Otherwise it should be 0.

13.5.4. Call of execution.

When the above preparations are made, the execution is called by transferring the first track of the translated program to location 264e4 and following and transferring the control to its first left instruction. The translated program will use the core store locations 0, 1023, and from e38 to, but not including, e37, where e38 and e37 are associated with a translator, see section 14. What other storage is used depends on the action of the particular Algol program.

13.6. OPERATOR CONTROL.

As described above the system is ready to be incorporated into a variety of surrounding driver programs. These will define the manner in which the various translation parameters are supplied from the user and the form of the operator control. Since it is expected that different installations will have different needs, a uniformity in these matters will not be attempted here. As a guide to the development of such driver programs one particular, simple one will be described in a separate publication of the Gier System Library.

14. PAPER TAPE FORM OF SYSTEM.

The programs of the Gier Algol 4 system are distributed in the form of a set of paper tapes coded in the version of Slip treating conditions. This representation leaves a number of options open, as described below.

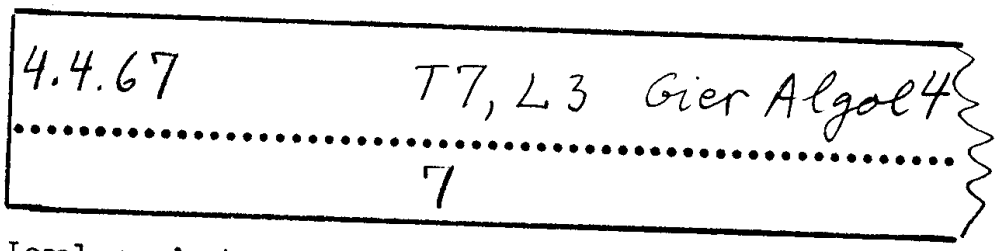
14.1. TAPE IDENTIFICATION AND CHECK.

In order to provide a safe way of making and identifying corrections to the system as they are required after the initial distribution, each tape of the system includes a version number. A particular version number is only used for one of the tapes of the set. Consequently a particular set of tapes is fully identified by the greatest version number of any tape of the set. During reading of the tapes by means of Slip it is checked that the greatest version number is consistent with the version numbers of all other tapes of the set.

Each tape of the system contains at the very beginning an identifying text of a form shown in the following example:

```
[4.4.67
                    T7, L3 Gier Algol 4
                    7]
```

Here L3 identifies the part of the system included, while 7 in the second line is the version number. It is recommended that all copies of tapes are marked clearly with precisely the text given within brackets, as for example:



Local variants of the system should be identified by additions to the version number consisting of / and any desired additional characters, like 7/KU23.

All tapes contain character check sums. These are based on the conventions described in section 7.1.4 and will be verified during input by the later versions of Slip (version number 28 or higher).

14.2. TRANSLATOR AND LIBRARY TAPES.

The part numbers and contents of the tapes belonging to the system are as follows:

Part	Contents
T1,L1	General pass administration, running system
T2	Pass 1, 2, 3.1,
T3	- 3.2, 4, 5.1, 5.2
T4	- 6, 9
T5	- 7, 8.1, 8.2, 8.3, 8.4
T6,L2	Library processor, part 1
T7,L3	Standard procedure library
T8,L4	Library processor, part 2
T9,L5,M1	Merger
P1	Process translator

The tapes are used either as a complete set or in certain selections. The set of tapes to be selected for any particular use is designated by one of the letters T, L, or M, and must be read into the machine in the order given by the following figure. The tape P1 may be read in whenever a finished translator is present on the drum. It contains various routines for output of the translator or parts of it.

The variations in the way the tapes are used arise because of the necessity that the information about the library of standard procedures, which initially is collected in the standard procedure library tape, is divided into several parts which must subsequently be merged into the text of the translator.

The result of the reading of a set of tapes is that certain parts of the system are placed in a known section of the drum, the subsequent use of it being left to the program on tape P1 or to the individual user. In particular the paper tape version of the translator (the transient translator) may be produced by tape P1.

The parameters defining the version of the system produced in a particular use of the tapes must be given at certain stages of the Slip-reading process, as indicated in the following annotated parts of the Slip texts of the tapes.

```
[7.6.67.                                                T1, L1 Gier Algol 4
                                                        1]

b c100, e100 ; Outermost block
             ;

[Definition of loading parameters
           Possible values     , meaning]

e14=90    ; Depending on available drum, first track for reading by slip.
e4=15     ;     10 < e4  < 17  , first core used by translator.
e20=1022  ; 1015e4 ≤ e20 ≤ 1022, last   -    -    -      -
e38=15    ;     10 ≤ e38 ≤ 200 , first core used by translated program.
e37=1022  ;    800 ≤ e37 ≤ 1022, last   -    -    -      -        -
          ;                     Note below: e37 = 1e37.
e27=0     ;   0, arrays in core, e18 and e40 must also be 0.
          ;   1,   -    - buffer.
e44=0     ;   0, translator medium is drum.
          ;   1,     -        -    - a buffer medium.
e18=0     ;   0, backing store is drum.
          ; 400,     -      -    -  disc with block length 400.
          ; 640,     -      -    -    -    -    -    -  640.
e40=0     ;   0, no tape stations, record handling procedures not included.
          ;n<15,  n  -      -        -        -        -      are     -
s         ; Now any or all of the above parameters may be redefined.

T9, L5, M1, first part of tape
b i=15, a30, b20, c20, e20  ;
d e4 = i                    ; first core location used during translation
I if e4 ≠ 15 then set e4
I if any of the following three parameters is not set, read in
I the necessary parameters
[e4: qq first track of system.39  ; set by tape T5
1e4: qq first track of library.39 ; set by tape L4
2e4: qq first track of working area.39 ; set by tape L4 to be the first
     free track following the library. 27 working tracks are needed]
d i = e4 + 3
```

The tapes may be used in four different ways, as follows.

A. Separate library.

Reading the tapes L1 to L4 will place the library on the drum, in a form which may be used in a later merging with a translator. At the end of the process the tracks occupied are given in the core store locations 1e4 and 2e4, as described above in the note on the first part of T9,L5, M1.

B. Total system from tapes.

Reading the complete set of tapes T1 to T9 will read a translator and a library and merge them to form a final system. The final system will be stored on the drum, starting in the first word of the track defined be e14. The storage of the parts of the system is described in a

table within the system itself, accessible through the first words of the system. These first words, considered in what follows to be placed in relative address 0 of the system, has the contents:

```
0:    qq  n , xxxx
1:    qq  relative address of segment table.9 + e4.19 +
          e27.20 + e44.21 f,
```

where n has already be described in section 13.2.6 and the segment table pointed to has one word for each separate segment of the system. These words have one of four formats, as follows:

Format 0; used only to describe the General Pass Administration, GP:

qq check.9 + words.19 + version number.29

Format 1, describing a new pass:

qq first core.9 + words.19 + pass no.24 + pass bits.29 + entry.39

Format 2, describing the next part of the current pass:

qq first core.9 + words.19 + 1.20 + entry.39 f

Format 3, describing a library segment:

qq words.19 f,

The parameters entering into these formats are: check: a check used internally; words: the number of words in the segment; first core: the first address in the core store in which the segment must be placed when used; pass no: the pass number; pass bits: (if change direction then 1 else 0).29 + (if backward pass then 1 else 0).28; entry: the address in the core store of first entry. The words of the segment table are arranged as follows:

Segment table

Relative address within table	Segment described	Format
0	General pass adm.	0
1	Pass 1	2
2	Pass 2	1
3	Pass 3.1	1
4	Std. identifiers	3
5	Pass 3.2	2
6	Pass 4	1
7	Pass 5.1	1
8	Pass 5.2	2
9	Library descriptions	2
10	Pass 6	1
11	Pass 9	1
12	Pass 7	1
13	Pass 8.1	1
14	Library programs	3
15	Pass 8.2	2
16	Pass 8.3	2

If the tape P1 is used to produce a transient translator the complete system is output on paper tape in two parts, the first containing the General Pass Administration and Pass 1, the second the rest of the segments. Each segment is punched as a number of spaces followed by the symbol a followed by 6 characters for each word in the segment thus:

qq char 6.6 + char 5.13 + ... + char 1.41

Between the output of the two parts the program stops, waiting for the operator to include suitable amounts of blank tape between the parts. It is restarted by typing BLANK.

C. Alter library from tapes.

A complete system, stored on the drum in precisely the form it was originally formed, but placed in any convenient place on the drum, may be combined with a new library, in the form of a set of tapes L1, ... , L5. The result of this process must always agree with the earlier version of the system with respect to the various parameters.

D. Alter library from drum.

This process is similar to C, except that the library must be provided in the form of a set of words stored on the drum, as produced by an application of process A. The reading only includes tape M1, Merger.

A survey of the uses of the tapes in the four cases and of the relevant possibilities of redefining the parameters is given in the following table, where letter X indicates that the possibility is relevant to the tape use in question.

	A	B	C	D
Place complete system on the drum			X	X
Place separate library on the drum				X
Call Slip	X	X	X	X
Start reading T1,L1, possibly redefine storage and mode parameters	X	X	X	
Continue Slip reading of tapes T2 to T5		X		
Continue Slip reading of tape T6, L2, possibly redefine first track of library	X	X	X	
Continue Slip reading of tapes T7,L3 and T8,L4	X	X	X	
Read program for further administration, written by user	X			
Read T9,L5,M1, possibly redefine e4		X	X	X
Set contents of e4			X	X
Set contents of 1e4 and 2e4				X

14.3. MODIFIED LIBRARY.

The paper tape L3 contains all information about the library of standard procedures and variables. Standard procedures may be added to the system or removed from it by modifying this tape. Since this will be of interest only to a few specialists, the description of the convention is given in a separate publication of the Gier System Library.

Appendix 1. EXECUTION TIMES.

Owing to the automatic administration of program segments (see section 11.3.) the execution time of an algorithmic constituent depends on the program loop structure and the number of variables declared at the time of execution. The times given below include an average segment administration time, such as it may be expected in loops which may be accomodated completely in the core store. Substantially longer execution times will result under the following circumstances: a) Frequent transfers of program tracks from drum are necessary. b) A major part of the execution time is spent in a loop with a cycle time of the order of 2 milliseconds or less and this loop happens to have been placed across a program segment transition by the compiler. The first of these calamities may only be remedied by using less core store for variables. It is unlikely to happen if arrays are stored in the buffer store. The second calamity may be cured by using the loop storage control described in section 11.4.

In any case the times given should be used only as a rough guide. Also, the difference between the times given here and those given for Gier Algol III do not always indicate a significant change of the system.

Except for some cases of expressions involving constants, each addition to the text of the program will add to the execution time. The execution time of an expressions may be found as the sum of the time taken to refer to each operand and the time of the operators. All times are expressed in milliseconds.

Operand reference times

Constant (see also below)	0.03
Simple variable, or formal called by value; the variable is declared	
in outermost block of program	0.00
in local block	0.00
in intermediate block	0.04

Subscripted variable	Arrays in core store		in buffer store	
Subscript check	Yes	No	Yes	No
1 subscript, A[ ]	0.7	0.4	0.6	0.3
2 subscripts, B[,]	1.1	0.8	1.0	0.7
3 subscripts, C[, ,]	1.5	1.2	1.4	1.1

Formal parameter called by name, specified integer, real or Boolean: the corresponding actual parameter is:	
constant	0.1
simple, the formal is used in expression	0.1
- - - - - as left part	0.3
other expression	3.2

Operators

To take care of conversions between integer and real types in operations with mixed types the operators round and float are generated internally as needed (cf. section 7.3). The examples are correct only if the operands are such which do not require execution time in themselves.

Monadic operators, i.e. operators having one operand

Negative, integer	-i	0.06
Negative, real	-r	0.11
round	round r	0.13
float generated internally		0.13
abs, integer	abs i	0.06
abs, real	abs r	0.12
entier	entier r	0.3
integer, real, Boolean, string operating on real		0.1
same, operating on integer, Boolean, or string		0.00
Not	-, b	0.14

Dyadic operators, i.e. operators having two operands

Plus or minus, integer	i - i	0.06
- - - , real	r - r	0.12
Multiplication, integer	i × i	0.35
- , real	r × r	0.2
Division	r / r	0.2
Integer divide	i : i	0.5
Modulo	i mod i	0.5
Power, integer exponent, square	r ↑ 2	0.2
- - - , cube	r ↑ 3	0.4
- - - , exp.=1	r ↑ i	3.4
- - - , - 10	r ↑ i	5
- - - , - 100	r ↑ i	7
- , real exponent	a ↑ r	6.5
Relational operators	a = r	0.1 - 0.3
And	p ∧ q	0.05
Or	p ∨ q	0.07
Imply	p => q	0.2
Equivalent	p = q	0.2
Shift, variable amount	b shift i	0.4
- , constant -	b shift 22	0.2
Assignment statement	a := b	0.1
Go to statement		
Simple, within current block	go to A; A:	0.8
To switch designator	go to s[i]	9.6
Call of declared procedure having an empty procedure body (procedure statement or function designator)		
No parameters	P;	5.1
1 parameter	Q(a);	5.6
2 parameters	R(a,b)	6.2
3 parameters	S(a,b,c)	6.6
If clause	if b then	0.2
Case clause: The time of one selection is greater when there are more complicated elements		0.8 - 3
For clause, each loop		0.4
Block with simple variables	begin real a; end	1.7
- - array declar.	begin array a[1:10]; end	3.2

Appendix 1. EXECUTION TIMES.

Translator evaluation of expressions involving constants

Operations involving only constant operands are performed during translation and thus do not contribute to the execution time in the following cases:

+ and - as monadic operators, or when giving a real result
× when giving a real result
/ , round, automatic conversions from integer to real type and vice versa, integer, real, Boolean, and string, in all cases.

The result of an operation performed during translation is again treated as a constant and may cause further operations to be performed during translation. Examples:
A[-2 + 6/2] is reduced to A[1] during translation
real := integer 30 1 is reduced to real := 1024.0
p - 3.4 - 5.6 is not reduced because the first operation contains p.
-3.4 - 5.6 + p is reduced to -9.0 + p
2 × 3.141592 is reduced to 6.283184
4 + 8 is not reduced because the result is of integer type.

Standard procedures of Algol 60

abs(x)	0.17	ln(x)	4.4
arctan(x)	5.3	sign(x)	1.7
cos(x)	4.5	sin(x)	4.8
entier(x)	0.4	sqrt(x)	4.9
exp(x)	4.4		

Standard procedures of Gier Algol

checksum	1.9	select	0.4
gier	0.1	system	25
kbon	0.11		

The following times refer to input from the RC 2000 paper tape reader, which completes the input of a character in 0.5 milliseconds. If a slower device is used the times may have to be increased. The times are expressed in terms of the number of characters read in, N. This must include every single character taken from the input medium, counting e.g. BLANK, UPPER CASE, and LOWER CASE.

		N=1	N=3	N=10	N=100
lyn, assuming no waiting for reader		0.22			
read general					
integer array	10.6 + 0.72 × N		13	18	83
real array	10.6 + 0.83 × N		13	19	94
read integer	2.7 + 0.63 × N		5	9	
read real	4.9 + 1.43 × N		9	19	
read string	11 + 0.95 × N		14	20	106

The following times refer to output to the line printer, which in this context accepts the characters as fast as they are produced. If a slower device is used the times may have to be increased. The number of characters produced as output is denoted N.

		N=1	N=10
write	6 + 2.6 × N		32
writechar, assuming no waiting for device,		0.5	
writecr, - - - - -		0.22	
writeinteger	3 + 1.6 × N		19
writetext	6 + 1.7 × N		23

The following times refer to a machine with backing storage on drum. The catalogue is 100 items large

cancel	400 to 700
reserve	300
where	200
get, put	6 + 21 × number of tracks transferred

Appendix 2. EXECUTION TERMINATION.

Any termination of a program execution which does not pass through the final end of the program will give a message on the output medium given as described in section 13.5.1. This message is typed in red and has the form:
<text> <line 1> - <line 2>, <relative track number>
The text is one of those explained below. Line 1 and line 2 are line numbers referring to the original Algol program text, the initial begin being in line 0 and relative track number gives the place in the translated program which give rise to the termination. The relative track number is 1023 for the last track of the program. The possible texts and the situation causing the execution termination are as follows:

array
: In array declaration the number of elements is negative or too large for capacity. In machines without buffer store the capacity for arrays in the buffer is zero.

case
: The value of the expression of a case clause is not positive or greater than the number of expressions or statements governed by the clause.

error 11
: In calling put or get the array has less than 40 elements or reaches outside of the backing store area, cf. 11.10.4.

error 12
: Standard procedure il or us is called with incompatible parameters.

error 13
: In calling system the array given as parameter does not have 40 elements.

exp
: Standard procedure exp is called with an argument greater than 354, causing the range of reals to be exceeded.

formal
: Assignment to formal name corresponding to an expression which is not just a variable is attempted.

index
: A reference to subscripted variable having subscripts outside the bounds of the corresponding declaration is made. The test for this situation is made only on the final address, not on the individual subscripts. Therefore the alarm will not always be made when the bounds are transgressed. Also the test may be suppressed when the translator is called, see section 13.2.6 parameter 10e4.

ln
: Standard procedure ln is called with a negative argument. Argument 0 does not call the alarm, cf. section 7.5.2.

mult
In multiplication of two integers the range is exceeded, cf. section 7.3.

spill
In arithmetic operation or during input the range of numbers is exceeded, cf. sections 7.3 and 10.11.3.

sqrt
Standard procedure sqrt is called with a negative argument.

stack
The capacity of the core store is exceeded by declarations, cf. section 11.2.

Appendix 3. MESSAGES FROM TRANSLATOR.

Most of the messages issued by the translator report logical flaws in the text supplied by the programmer, and great pains have been taken in the design of the system to make the error detection as complete as possible. There are, however, a few classes of such errors which are known to pass undetected. These are: (1) Use of local quantities in array bound expressions, cf. section 5.2.4.2. Such errors will lead to a use of bounds of unknown magnitude. Often this will be detected immediately at block entry by the array exceeding the capacity of the machine. In many other cases the error will be detected as a bounds-exceeded error when the elements are used. (2) In a procedure call of a formal procedure the actual parameters do not match the corresponding formals with respect to kind and type or the numbers are different. This may cause completely unpredictable reactions at execution time. (3) The test of the subscripts against the array bounds is made on the final address only, if at all (cf. appendix 2, index). (4) Overflow in arithmetic operations on integers is not normally detected, cf. section 7.3. (5) The second operand of shift goes outside the range -512 to 511. Each of the last three classes of errors will cause wrong results, but the control of the program will remain intact. (6) Machine code written by the programmer may cause any unpredictable reaction.

Unless otherwise noted below all messages are associated with a translation pass and a line. The pass number is output prior to the first message from each pass as explained in section 13.2.7. Each message is typed in red and has the form:

line <line number> <message text>

where the line number refers to the Algol program text, the initial begin being in line 0. The possible message texts and their meaning in each pass are given below. An underlining is not part the text but indicates that the translation is terminated immediately.

BEFORE PASS 1

ready

First part of a translator on paper tape has been read and is ready for the Algol text. Type a BLANK to start the translation.
No pass or line number.

ALL PASSES

program too big

The working area provided for the translation according to 13.2.4 and 13.2.6 is insufficient for the Algol program text.

pass sum

The built in checksum for the next pass or phase does not agree.

pass medium

Transport error during loading of the next pass on phase.

PASS 1

character
The Algol text contains an illegal character, see section 7.1.3.2.

off
Character PUNCH OFF leads to skip of text, section 7.1.6.

on
Character PUNCH ON ends skip of text, section 7.1.6.

comment
The delimiter comment or message is not preceded by begin or ;

)<improper>
The construction)<letter string> is not followed by :(

code length
The limits on code length exceeded, cf. section 12.1.

≮ in string
The symbol ≮ is met inside a string. This is not an error, but is likely to be unintended.

compound
The input has a string of which some of the first, but not the following, characters represent some of the first characters of a compound symbol, cf. section 7.1.2.

type in
A copy call requests the continued Algol text from the typewriter, cf. section 13.2.1.

pause
END CODE is found in the Algol text, cf. section 7.1.5.

copy
The lookup program is missing or the init medium program is missing or the copy source does not point to an item in the catalog containing a string of characters, cf. section 13.2.1.

copy medium
Transport error during a copy from a buffer medium

copy overlap
The next track of the Algol text on the drum is destroyed by the translator.

sum
Failure of the input check sum, cf. section 7.1.4.

string
The compound symbol ≮ is followed neither by < nor by a layout, cf. section 9.5.

stack
Too many copy levels are called, cf. section 13.2.1.

passes
Second part of the translator on paper tape must be readied for input. Type a BLANK to start it.
No pass or line number.

PASS 2

identifier overflow
: The program uses too many or long identifiers. Remedy: use the block structure to reduce the number of different identifiers. The maximum capacity is 511 short identifiers.

pattern
: A bit pattern has a wrong structure or the word or part pattern capacity is exceeded, cf. section 7.7.1.

PASS 3

zero
: The selector of the floating point zero treatment is in the wrong position. Reset selector and type BLANK.

std proc format
: The information found in the library does not conform to the proper library format

double std proc
: The library contains the same identifier twice.

-delimiter
: Two operands, i.e. identifiers, numbers, logical values, strings, or compound expressions within parentheses, follow each other.

operand
: An operand appears in a wrong context or is missing.

delimiter
: The delimiter structure is impossible.

-operand
: Operand is missing at end of construction

termination
: Parentheses, brackets, or bracket-like structures do not match.

head
: Erroneous structure or identifier match in procedure heading. Only one head message is given for one heading. This gives the line number of the first symbol of the procedure body.

const.
: Error of structure associated with one of the constant operands, i.e. number, pattern, layout, string, true, or false.

stack
: The nesting of begin-s, parentheses, etc. exceeds the capacity of the translator.

PASS 4

begin ends
The nesting of begin-end pairs exceeds capacity.

indices
The number of subscripts of an array exceeds capacity.

stack
The capacity of the stack for collecting the descriptions of simultaneously declared identifiers is exceeded. Remedy: Use the block structure to avoid the simultaneous existence of many quantities.

PASS 5

undeclared
An identifier is not declared

+ decl.
An identifier is declared two or more times in the same block. The message appears at each place of declaration.

stack
The capacity of the stack recording redeclarations of identifiers is exceeded.

std.procs.
The number of standard procedures used by the program is too large for capacity

PASS 6

call
A procedure call has an incorrect number of parameters

subscripts
A subscripted variable has an incorrect number of subscripts

type <clue>
Error of type or kind of operand with respect to the operator context. The clue gives further information about the error, by identifying the context of the error according to the following table:

128 <error> ; or <error> end or <error> else

131 if <error> then	157 <error> ∨	183 <error> ;
132 then <error> else	158 <error> ≠	184 <error> ,
134 else <error>	159 <error> =	185 for <error> :=
137 go to <error>	161 <error>)	186 until <error> ,
139 <error> step	162 <error> do	187 while <error> ,
140 step <error> until	163 until <error> do	188 <error> while
143 <error> mod	164 while <error> do	189 := <error>
144 <error> +	167 case <error> of	190 := <error> :=
145 <error> -	168 ,<error>)	191 <error> :=
146 <error> ×	169 <error> ,	208 <error> (
147 <error> /	173 if <error> then	209 <error> [

148 <error> :
149 <error> ↑
150 <error> shift
151 <error> , or]
152 <error> , or]
155 <error> ;
156 <error> ∧
175 case <error> of
177 <error>)
178 <error>]
179 <error>]
180 <error> ,
181 <error> ,
182 <error> ,
214 <error> (
492 <error> <
493 <error> ≤
494 <error> =
495 <error> >
496 <error> ≥
497 <error> ≠

stack
The capacity of the stacks recording operands and operators being translated is exceeded.

PASS 9

number
Error associated with numeral in machine code.
syntax
The machine code does not conform to the restricted Slip syntax.
addr.
Error associated with Slip name or address.
code head
Error in code head
code size
Code exceeds the admissible number of words
undef. <Slip name>
The Slip name given is not defined.
sorry
The translation is terminated because of errors.

PASS 7

spill
Range of numbers exceeded in arithmetic expression containing constant operands.
stack
The capacity of the stack recording operands being translated is exceeded.

PASS 8

stack
The capacity of the stack of internal program references is exceeded.
case too big
A case clause governs too many expressions or statements, or a switch declaration has more than 34 designational expressions.

Appendix 4. ENVIRONMENT DESCRIPTION

The environment description is available to the Algol program through the procedure system (section 8.2) and has the form of an array of 40 Boolean elements. In the following explanation of the format of the description it is assumed to have been transferred to a Boolean array D[1:40].

The description consists of a number of separate parameters, packed into the elements of the array. It holds generally that the parameter value 0 is not used to indicate positive information, but is reserved to indicate that the parameter in question is not used in the version of description at hand. The other extreme of the range of a parameter is similarly reserved for the unlikely case that the capacity originally assigned to a parameter is later found to be insufficient. When this value is found it indicates that further information about the parameter value may be found elsewhere in the description.

The description consists of two elements, D[1] and D[2], in fixed format, describing the central machine, and additional words each describing a part of the peripheral units. The fixed format description consists of the following:

Bits in D[1]	Values and their meaning
0 - 3	Primary backing store, tracks of 40 words. (1): 1 drum, 320 tracks; (2): 2 drums, 640 tracks; (3): 3 drums, 960 tracks; (4): disk file, 9600 tracks, group selection and gg-instruction active.
4 - 6	Buffer store. (1): None; (2): 4096 words, il and us instructions active.
7 - 9	Sense busy instruction. (1): None; (2): il 256+channel active.
10 - 12	HP button and by-register. (1): No HP button, by has positions 2 to 9; (2): HP button, inhibition by by-position 3, by has pos. 2 to 9; (3): HP button, inhibition by by-position 0, by has pos. 0 to 9; (4): HP button tied to interrupt system.
13 - 14	by-position 3 to 6 assignment buttons, i.e. manual control of output medium. (1) Not available. (2) Available.
15 - 17	Output of 8-bit character by sy-instruction. (1) Not possible. (2) Achieved by adding 128 in by-register.

18 - 39 and D[2] are reserved for later extensions. In present version they must all be 0.

The following words describe each peripheral unit, using one word for each. The words are ordered in magnitude. The format of each word depends on the value of the first 10 bits, but mostly conform to the following arrangement:

Bits 0 - 9 Identity of unit
10 - 19 Associated value of by, to be used in select
20 - 37 Other description as given below
38 - 39 (1): The unit is permanently coupled to the machine.
(2): The use of the unit depends on the setting by the operator of a manual switch, other than the by-position assignment buttons.

Paper tape readers, identity 1 to 20.

(1): Facit; (2): RC 2000. If the by-value is given as 0 or 4, the reader can only be used with this value and will stop on reading a character of even parity. If the by-value is given as 3 or 7, either that value may be used, with no stop on even parity, or that value minus 3 may be used to cause stop on even parity.

Paper tape punches, identity 21 to 40.

(21): Facit. The by-value is given. For an additional possibility, note D[1] bit positions 15 - 17.

Line printers, identity 41 to 60.

(41): Series 4, 1 line buffer; (42): Series 4, 2 line buffers; (43): Series 5. Bits 10 - 19 give the associated by-value. In machines equipped with by-position 3 to 6 assignment buttons, i.e. having D[1] bits 13 - 14 equal 2, this by-value assumes that the buttons along the diagonal are depressed. Bits 38 - 39 indicates the coupling to the machine, as described above. Bits 20 - 37 describe some of the details of the character set as follows:
Bits 20 - 22 = 1 indicate that the character set may be thought of as defined from a basic set, given in the following table, with certain variations defined by the following bits, as given in the table of variations.

Basic character set, bits 20 - 22 = 1

A blank position indicates that nothing is defined about the character in that position. An integer > 20, or a pair of integers, like 23-24, refer to the specification of variations in the following bit positions, as fully explained below.

	Lower	Upper
0	BLANK	
1	1	25
2	2	×
3	3	/
4	4	=
5	5	;
6	6	[
7	7	]
8	8	(
9	9	)
10		
11		
12	26	
13	23-24	Å
14	27	
15	%	&

	Lower	Upper
16	0	28
17	<	>
18	23-24	S
19	23-24	T
20	23-24	U
21	23-24	V
22	23-24	W
23	23-24	X
24	23-24	Y
25	23-24	Z
26		
27	,	¤
28		
29		
30	29-30	
31		

	Lower	Upper
32	-	+
33	23-24	J
34	23-24	K
35	23-24	L
36	23-24	M
37	23-24	N
38	23-24	O
39	23-24	P
40	23-24	Q
41	23-24	R
42	31	
43	23-24	Ø
44	32	
45	33	
46		
47		

	Lower	Upper
48	23-24	Æ
49	23-24	A
50	23-24	B
51	23-24	C
52	23-24	D
53	23-24	E
54	23-24	F
55	23-24	G
56	23-24	H
57	23-24	I
58	LOWER CASE	
59	.	:
60	UPPER CASE	
61		
62		
63		
64	CARRET	

Variations on basic character set, valid when bits 20 - 22 = 1; Bits 23 - 24: (1) The character in lower case is the same as in upper case. (2) The character is the corresponding lower case one.

Bits(s)	Character value described	Bits = 1	Bits = 2
		Lower Upper	
25	1	1 V	
26	12	23-24 Ü	
27	14	|	
28	16	Ō Λ	
29-30	30	SET POSITION	TAB
31	42	FORMFEED	
32	44	* ,	
33	45	PAPERTHROW	

For each of the parameters in positions 25 to 33 the value 0 indicates some other effect of the character value.
Bits 34 - 35: Effect of character values 65 to 80:

(0) Other than as (1) or (2).
(1) 65-79: Paper throw controlled by format control tape. 80: printing without paper motion.
(2) 65-79: Paper throw controlled by format control tape. 80: printing with double line feed.

With bits 20 - 22 not equal to 1 nothing is specified here about the character set. The values 2, 3, ... , 7 may be used later to specify other specific character sets.

Plotters, identity 61 to 80.
(61): Calcomp. The type number is given in bits 20 to 30. (62) Moseley Model 2D 3M.

Card readers, identity 81 to 100.
(81): Modified Bull D3. (82): CDC 9200.

Tape drivers, identity 101 to 120.
(101): Ampex TM 7. (102): CDC 606.

Caroussel, identity 121 to 140.
(121): Facit ECM 64

Disk file, identity. 141 to 160.
(141): Anelex model 80, replacing drum. (142): Anelex model 80, transferring to buffer store. (143): CDC 9433.