THE NU ALGOL

PROGRAMMING SYSTEM

FOR

UNIVAC 1107/1108

PROGRAMMERS GUIDE

AND REFERENCE MANUAL

COMPUTING CENTRE NTH

TRONDHEIM NORWAY

TAPIR

DECEMBER 1969

PREFACE

The system described herein was initiated with two objectives:

- to provide ALGOL-users with a reasonable efficient and reliable programming tool.
- to serve as an adequate base for the implementation of the SIMULA-67 language.

For practical reasons it became necessary to be compatible with UNIVAC's old ALGOL system which thus served as the detailed definition of the source language. However, in a few places the compatibility has been sacrificed to achieve a more efficient and reliable implementation. From a pragmatical point of view these are regarded to be of no significance for most users.

This manual is a first edition and is made in a lose leaf form to make later corrections and supplements easy.

ACKNOWLEDGEMENT

This system was designed and implemented as a joint effort of the Norwegian Computing Centre, Oslo and the Computing Centre at the Technical University of Norway, Trondheim. The design of the storage allocation scheme for data and program is done mainly by prof. O.J. Dahl, Mr. S. Kubosch, Mr. B. Myrhaug and Mr. K.S. Skog. The implementation of the runtime system is by Mr. Kubosch. Mr. B. Myrhaug made the design of the I/O section which was implemented by Mr. R. Kerr and Mr. B. Meldrum.

The design and implementation of the compiler is mainly by Mr. O. Meland, Mr. K. Rekdal and Mr. K.S. Skog.

Mr. Kubosch made the compiler interface to the EX-2 system.

Mr. B. Meldrum wrote the first draft of this manual. In addition on part time and/or in shorter periodes Mr. A.O. Østlie, Mr. N. Bull, Mr. D. Belsness, Mr. H. Nordvik, Mr. A. Øverby and Mr. K. Sundnes has been involved with the project.

The final corrections and proffreading of the manual was done by Mr. T. Noodt and Mr. K. Rekdal.

I wish to express my gratitude to all mentioned here and in addition to their wives and children who undoubtedly has suffered in periodes when teory and code did not really match. Last but not least a particular thanks to Mrs. L. Aasheim and Miss M. Sundet for their patient interpretation of many cryptic manuscripts.

The project has been superviced by Knut Skog.

CONTENTS

UNIVAC 1107/1108 NU ALGOL

1 INTRODUCTION *)

1.1 General

NU ALGOL is a language for communicating scientific and data processing problems to the UNIVAC 1107/1108 computers. The basis for this language is the "Revised Report on the Algorithmic Language ALGOL 60" (P. Naur (ed.), Regnecentralen, Copenhagen 1962). This implementation of ALGOL 60 is very close to that of the report. It's one significant omission is the omission of all own variables. It's significant additions include three new types STRING, COMPLEX and REAL2 as well as the allowing of external procedures written in machine language or FORTRAN and the definition of a versatile input/output system.

NU ALGOL is compatible with UNIVAC 1107/1108 ALGOL with the few exceptions noted in appendix E - "Differences between NU ALGOL and UNIVAC 1107/1108 ALGOL". The major differences between the two are the actual method of compilation, the extended input/output facilities, and a major improvement in both runtime and compiletime security and speed.

1.2 Scope and Format of this Manual

Scope

The layout of this manual has been designed to provide fast reference to all features of the language so that those familiar with ALGOL may look up points easily. At the same time, many examples have been inserted to allow beginning programmers to become familiar with the features of the language.

No attempt has been made to illustrate all the constructions possible, however, appendix F contains a complete syntax-chart for NU ALGOL.

*) References

This introduction is based on material contained in the UNIVAC 1107 Programmer's Guide.

Format

Although the ALGOL report cited above uses underlining to delineate basic symbols, this manual does not. All explanations and examples give the basic symbols as they would be found on printer output from the computer - that is in upper case letters with no underlining.

In describing forms of constructions (syntax) the bracket pair < and > are used to isolate the constructions under definition. For a complete and unambigious definition of syntax see appendix G.

1.3 The NU ALGOL Compiler

The NU ALGOL compiler is a program which accepts statements expressed in ALGOL and produces programs for the UNIVAC 1107/1108 computers.

An ALGOL program is a sequence of statements written in the ALGOL language. These are translated by the compiler into the language of the computer: machine language. The ALGOL statements are called the source code, and the translated statements are called the object code. The compiler itself is a program written in machine language and is called the UNIVAC NU ALGOL Compiler. While translating the ALGOL statements, the compiler looks for errors, and reports these back to the programmer.

The compiler operates in four passes. Upon successful compilation, the object code can be read into the main storage and executed. Activities that occur during compilation are sometimes referred to as compile-time activities; for instance, compile-time diagnostics. The execution phase is referred to as run-time.

1.4 Differences between ALGOL 60 and NU ALGOL

1.4.1 Extensions to ALGOL 60

a) The addition of STRING and STRING ARRAY variables has been made to enhance the value of ALGOL as a data processing language.

b) The addition of the arithmetic types COMPLEX and REAL2 has been made to enhance the value of ALGOL to scientific users.

c) XOR has been added to list of logical operators.

d) EXTERNAL PROCEDURE declarations have been implemented to allow easier programming of large problems and the building of program libraries.

e) Input and output routines have been defined along with FORMAT and LIST declarations to be used by them.

f) A compact form for GO TO and FOR statements has been provided.

g) Variables are zeroed upon entry to a block so that initialization statements are not required.

h) The controlled variable of a FOR statement has a defined value when the statement is terminated by exhaustion of the FOR-list.

1.4.2 Deletions from ALGOL 60

a) The following limitations have been imposed.

Identifiers are unique only with respect to their first 12 characters.
Identifiers may not contain blanks.
Numbers may not contain blanks.
Certain ALGOL words may only be used in a specific context.

b) own variables are excluded.

c) Numeric labels are not allowed.

d) The comma is the only delimiter allowed in a procedure call.

e) The result of an integer raised to an integer power is always of type REAL.

f) All the formal parameters of a procedure must be specified.

g) In a Boolean expression all operands are not evaluated when this is not necessary for determining the result.

2 BASIC INFORMATION

2.1 Basic Symbols

The following symbols have meaning in NU ALGOL:

Simple symbols

The letters	A - Z
The digits	0 - 9
The logical constants	TRUE FALSE
The ALGOL symbols	
Arithmetic operators	+ - / *
Special characters	= ( ) , $.
	; & < > ' | | :

A space (blank) symbol

Compound symbols

Some multiples of characters are given meaning as if they constituted a single character:

//	(integer divide)
**	(exponentiation)
&&	base 10 scale factor for double precision constants
:=	replacement (instead of =)
..	colon (same as :)

A set of reserved words such as:
BEGIN END IF THEN etc.
A complete list is given in 2.5.

For details on card code and character set, see appendix A.

2.2 Identifiers

Purpose

Identifiers (apart from those mentioned in 2.5) have no inherent meaning, but are names that the programmer chooses to use to refer to various objects (operands, procedures, labels etc.).

Rules for identifiers

a) An identifier is combination of characters taken from the set of letters (A - Z) and the set of digits (0 - 9).

b) The first character of an identifier must be a letter.

c) Although any number of characters may be used to make an identifier, only the first 12 uniquely specify the identifier.

d) It is often easier to read the program if the identifier is a mnemonic.

Examples:

i)	A	P060	Z1Z4	KAF1
ii)		NONLINEARRESIDUE		
		NONLINEARRESULT		

are considered identical because their first 12 characters are the same.

2.3 Form of an ALGOL Program

ALGOL programs are made up of one or more blocks. The concept of blocks is treated in section 6. In brief, an ALGOL program containing only one block has the following form:

```
BEGIN
  <Declarations>$
  <Statements>
END$
```

Declarations are described in Section 3.
Statements are fully treated in Section 5. Briefly the following are true.

a) Statements are orders to perform one or more computations or input/output operations.

b) Statements are separated from each other by the symbol $ or the symbol ; (Either may be used).

c) Exit from a block must be through the final END or through a jump to a label in an enclosing block.

2.4 Layout of an ALGOL Program on Cards

The source code to the compiler must come initially from punched cards. The following rules should be followed.

a) Only columns 1 through 72 are read for information.

b) Columns 73 through 80 may be used for any purpose.

c) The compiler considers that there is space between column 72 of one card and column 1 of the next card except in strings.

d) One or more statements may be placed on one card.

e) The program text should be arranged to make the program readable and easy to change.

2.5 Special Identifiers

2.5.1 Reserved Identifiers

The following sets of characters have special meanings and may not be used as identifiers.

ALGOL	GOTO	SLEUTH
AND	GTR	STEP
ARRAY	IF	STRING
BEGIN	IMPL	SWITCH
BOOLEAN	INTEGER	THEN
COMMENT	LABEL	TO
COMPLEX	LEQ	TRUE
DO	LIBRARY	UNTIL
ELSE	LIST	VALUE
END	LOCAL	WHILE
EQIV	LSS	XOR
EQL	NEQ	
EXTERNAL	NOT	
FALSE	OFF	
FOR	OPTION	
FORMAT	OR	
FORTRAN	PROCEDURE	
GEQ	REAL	
GO	REAL2	

2.5.2 Standard Procedure Identifiers

The following identifiers may be used without explicit declarations for calling standard procedures.

ABS	LINEAR
ALPHABETIC	LN
ARCCOS	MARGIN
ARCSIN	MAX
ARCTAN	MIN
CARDS	MOD
CBROOT	NEGEXP
CHAIN	NORMAL
CLOCK	NUMERIC
COMPL	POISSON
CORE	POSITION
COS	PRINTER
COSH	PSNORM
DISCRETE	RANK
DRAW	RANDINT
DRUM	RE
DRUMPOS	READ
DOUBLE	REWIND
ENTIER	REWINT
EOF	SIGN
EOI	SIN
ERLANG	SINH
EXP	SQRT
HISTD	TAN
HISTO	TANH
IM	TAPE
INT	UNIFORM
KEY	WRITE
LENGTH	

These identifiers may however be redeclared for other use.

For details on standard procedures see section 7.4.

3. DECLARATIONS

3.1 Introduction

Purpose

Declarations are used to inform the compiler that identifiers have certain attributes. A declaration for an identifier is valid for one block, inner blocks inclusive.

Rules for identifiers

1. All identifiers used in a program, except standard procedure identifiers, must be declared.
2. In a block (see section 6) an identifier may be declared only once.

Type declarations

Variables are names which are said to possess values. These values may in the mathematical sense be integers, real numbers, or complex numbers. In addition there are the possibility of the truth values TRUE or FALSE. All these are different types of values. A variable of a certain type can only possess certain values partially according to the rules of mathematics and partially because of hardware limitations.

In this manual the symbol <type> will be used to mean that this symbol can be replaced with one of the following ALGOL types which then impose the limits shown.

<type>	Value	Limits
INTEGER	Integral values:	[-34359738367, +34359738367]
REAL	Real values:	$(-3.37\times10^{38}, -1.48\times10^{-39})$, 0, $(1.48\times10^{-39}, 3.37\times10^{38})$ Up to 8 significant digits
BOOLEAN	Truth values:	FALSE, TRUE
COMPLEX	Complex values:	Same limits as for REAL since the real and imaginary parts are treated as two separate real numbers.

<type>	Value	Limits
REAL2	Real values:	Same limits as for type REAL but up to 16 significant digits.
STRING	Alphanumeric characters	Any character in the UNIVAC 1107/1108 character set.

Initial values of simple variables

All variables declared in a block are initially set when the block is entered. For variables of type INTEGER, REAL, REAL2, and COMPLEX the initial value is zero (0). For BOOLEAN variables the initial value is FALSE. For STRING variables the initial value is a sequence of blanks.

3.2 Declaration of Simple Variables

Purpose

A simple variable is a non-subscripted name for a value of a given type.

The declaration of a simple variable defines the type of value the identifier for that variable may assume.

Examples:

```
INTEGER   A $
REAL      B1,C2,D $
BOOLEAN   RIGHT,ANSWER $
COMPLEX   ROOTS $
REAL2     BIGNUMBER,EVENBIGGER $
STRING    LETTERS (25) $
```

Form

<type><list of identifiers>$

<type> is defined in 3.1.

List of identifiers means one identifier (see section 2.2) or several identifiers separated by commas.

The declaration ends with the character $ or ;

3.2.1 Declaration of a Simple String.

Purpose

The declaration of a simple string variable provides a means of storing and referring to a collection of alpha-numeric characters in Fielddata code by the use of a single identifier.

Form

STRING <identifier> (<string part>)

Identifier is defined in 2.2.

String part is an integer expression (in the outer-most block of a program, an integer constant), whose value is the maximum number of characters to be kept in the string.

In a substring declaration string part may also be a list of integer expressions and string declarations separated by commas. (See sec. 3.2.2 below)

Examples:

```
        STRING      S1 (25) $
        STRING      S2 (14), CHARAC (22), LTRS (4) $
```

In an inner block also:

```
        STRING      CHARS (N) $
```

3.2.2 Declaration of a Substring.

A substring is a part of main string and has the same properties as a string.

A substring is declared by placing an identifier and a string part in the string part of the main string.

The length of the main string is then the sum of the lengths of its substrings plus any other lengths specified.

Note: The length of a string may not be specified by the call of a type procedure as this will be taken as a substring declaration. If the type procedure and the main string are declared in the same block, this ambignity will give the error message "DOUBLE DECLARATION".

Examples:

```
STRING SOUT (SIN1(20),SIN2(42))$
```

SOUT has a length of 62 characters.

SIN1 is a substring of length 20 and is the same as characters 1 through 20 of the main string SOUT.

SIN2 is a substring of length 42 and is the same as characters 21 through 62 of the main string SOUT.

```
STRING LTRS (10,NUMBS(12),4,CHRS(6))$
```

LTRS has a length of 32.

NUMBS has a length of 12 and is the same as characters 11 through 22 of the string LTRS.

CHRS has a length of 6 and is the same as characters 27 through 32 of the string LTRS.

3.2.3 Storage required by Simple Variables.

The memory of the UNIVAC 1107/1108 computers is divided into "words" each consisting of 36 bits.

Each identifier reserves a number of words depending on its type.

TYPE	NUMBER OF WORDS
INTEGER	1
REAL	1
BOOLEAN	1
COMPLEX	2 - one for real part - one for imaginary part
REAL2	2 - to allow the carrying of more significant digits
STRING	The integer value given by ENTIER ((Length + start pos. + 11)/6) where start position goes from 0 to 5 and length is the number of characters in the string.

3.3 Declaration of Subscripted Variables (array).

Purpose

An array is a set of variables each of which can be accessed by referring to an identifier with one or more subscripts.

Each member of the set has all the properties of a simple variable.

The declaration of an array defines the type of value each member of the array may assume, the number of subscripts required, and their limits.

Form

<type> ARRAY <array list>$

a) Type is defined in 3.1. If type is omitted, the type REAL is assumed.

b) Array list is a list of array segments, which have the form

<list of identifiers> (<bound pair list>)

A bound pair list consists of one bound pair or several bound pairs separated by commas.

A bound pair has the form

<aritmetic expression >:<arithmetic expression >

Section 4 defines arithmetic expression.

Note: In the outermost block the aritmetic expression can only be a constant

Examples:

```
INTEGER ARRAY     AI (0:25) $
REAL ARRAY        AR (1:3,1:3) $
COMPLEX ARRAY     AC (-2:20),AD,AE(14:24) $
BOOLEAN ARRAY     BA,BC,BD(0:5),BE(1:4) $
REAL2 ARRAY       K1,K2,KL,KF(-1:10) $
```

In an inner block also:

```
INTEGER ARRAY     A1 (N:N*4) $
```

3.3.1 Rules for Array Declarations.

a) Each bound pair defines the values the corresponding subscript may take. In NU ALGOL, the number of subscripts is limited to 10.

b) In a bound pair, the first arithmetic expression is called the lower bound. The second arithmetic expression is the upper bound. The lower bound must always be less than or equal to the upper bound.

c) The arithmetic expressions must be of type INTEGER or of a type which can be converted to INTEGER (REAL,REAL2).

3.3.2 Meaning of Array Declarations.

a) The meaning of an array declaration can best be explained by examples. An array declaration with one subscript position such as

```
REAL ARRAY A(0:10)$
```

declares 11 REAL subscripted variables:

A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8),A(9),A(10)

An array declaration with two subscript positions such as

```
ARRAY XY(-2:1,1:3)
```

declares 12 REAL subscripted variables:

```
XY(-2,1)    XY(-2,2)    XY(-2,3)
XY(-1,1)    XY(-1,2)    XY(-1,3)
XY( 0,1)    XY( 0,2)    XY( 0,3)
XY( 1,1)    XY( 1,2)    XY( 1,3)
```

Note that the use of a subscripted variable consumes substantially more computer time and program space than the use of a simple variable.

b) If several identifiers are followed by only one bound pair list then these identifiers each refer to an array with the number of subscripts and the bounds given in that bound pair list.

Example:

COMPLEX ARRAY CAD,CM,KF(4:20) $

This declaration defines three arrays each of type COMPLEX, with 17 members and with a lower bound of 4 and upper bound of 20.

Note that all these arrays occupy different areas of storage.

3.3.3 Declaration of a String Array.

Purpose

Subscripted STRING variables may be declared using the STRING ARRAY declaration. This gives the user a possibility of choosing among different strings by means of appropriate subscripting.

Form

STRING ARRAY <identifier>(<string part>:<bound pair list>)$

An identifier is defined in 2.2.
The term string part is defined in 3.2.
The term bound pair list is defined in 3.3.

Rules for string array declarations

A **string array** declaration must obey the rules for both string declarations and array declarations with the exception that each identifier must be followed by

(<string part>:<bound pair list>)

even if all characteristics are the same for the string arrays being declared.

Examples:

```
STRING ARRAY   SAX(14:0:5,1:4)$
STRING ARRAY   SAK(2,LAK(16):20:31)$
STRING ARRAY   KAS(KAL(2),4,KAT(20):-2:4,1:2)
STRING ARRAY   MEL(10:0:5),MELT(10:0:5)$
```

3.3.4 Meaning of String Array Declarations.

The meaning can best be shown in an example:
The declaration

```
STRING ARRAY L(2,M(5):0:3,1:2)$
```

defines 8 strings each of length 7:

```
L(0,1)        L(0,2)
L(1,1)        L(1,2)
L(2,1)        L(2,2)
L(3,1)        L(3,2)
```

and the 8 substrings of length 5

```
M(0,1)        M(0,2)
M(1,1)        M(1,2)
M(2,1)        M(2,2)
M(3,1)        M(3,2)
```

3.4 Other Declarations.

The following special declarations are decribed in the sections shown.

Declaration	Section
FORMAT	8.6.3
LIST	8.7.2
EXTERNAL PROCEDURE	7.3.2
PROCEDURE	7.1.2
LABEL	4.6.2
SWITCH	4.6.3

4 EXPRESSIONS.

4.1 Introduction.

An expression is a rule for computing a value, or a destination. There are 4 kinds of expressions: arithmetic, boolean, string, and designational. The constituents of these expressions, except for certain delimeters, are operands and operators. The operands my be constants, variables, or type procedure calls. The operators may be arithmetic, relational, boolean, and sequential.

Operators cause certain actions to be performed on the operands.
Certain operators may only be used in certain types of expressions.

Parentheses are used as in algebra to group certain operators and operands and thus determine the sequence of the operations to be performed. Parentheses have a special meaning in conditional expressions.

4.2 Arithmetic Expressions.

4.2.1 Meaning.

An arithmetic expression is a rule for computing a numeric value. A constant or a simple variable is the simplest form of an arithmetic expression. In the more general arithmetic expressions, which include conditions (if clauses), one out of several simple arithmetic expressions is selected on the basis of the actual values of the Boolean expressions.

4.2.2 Types of Values.

An arithmetic expression may produce a value with one of the following types (see section 3.2).

INTEGER
REAL
REAL2
COMPLEX

4.2.3 Arithmetic Operands.

a) Arithmetic Constants

The type of a constant depends on the form in which it is written. No blanks are allowed in a constant.
The following rules apply.

Type of Constant	Rules for Formation	Examples
INTEGER	A string of 11 or fewer digits possibly preceded by a '+' or '-' (see also sec. 3.1)	70 -204 0 + 0 - 25
REAL	1. A string of 8 or fewer digits with a decimal point within the string or at either end and possibly preceded by a '+' or a '-'	1.2 .1 -0.111 75.333333 +40.0 +1.
	2. A power-of-ten symbol (&) followed by an integer indicating the power, and possibly preceded by a '+' or '-'	+&7 &-2 &+6 -&-1
	3. An integer or a real number of type (1) followed by an exponent of type (2)	1&6 1.0&6 -17.446&-3 +6.&17
REAL2	1. A number of the same form as REAL types (1) or (3) but having between 9 and 16 significant digits.	1.2000127211 -203456789.12 1.031462873&-22
	2. A number of the same form as REAL types (2) or (3) but using the symbol '&&' to mean power-of-ten	1.0&&2 4&&0 +3.1629&&-4 0.0&&0

Type of Constant	Rules for Formation	Examples
COMPLEX	Two constants of the form for REAL or INTEGER separated by a comma and enclosed within the symbols '<' and '>' where the first constant represents the real part and the second the imaginary part of the complex constant.	<+7.0&-2,-2> < 1.0, 0.0> <-2, -1> <2.0, -1>

Notes 1&6 or 1&&6 means 1×10^{6} or 1000000.0
3.1629&&-4 or 3.1629&-4 means 3.1629×10^{-4} or 0.00031629.

b) Arithmetic variables

Arithmetic variables are those variables which have been declared to have one of the types

INTEGER
REAL
REAL2
COMPLEX

An arithmetic variable may be simple or subscripted (that is, an element of an array).

c) Arithmetic Type Procedures

The declaration of a type procedure is described in section 7.2.

In an arithmetic expression, procedures declared to have the following types may be used:

INTEGER
REAL
REAL2
COMPLEX

All standard procedures (e.g. SIN, COS, ENTIER, LN, etc.) which return a value of type INTEGER, REAL, REAL2, or COMPLEX may also occur in arithmetic expressions.

4.2.4 Arithmetic Operators.

a) The Operators.

The following arithmetic operators are defined in NU ALGOL and have the meanings indicated below:

Operator	Meaning
+	If not preceded by an operand then monadic plus - that is the following operand has its sign unchanged. If preceded by an operand and followed by an operand then the algebraic sum of the two operands is to be calculated.
-	If not preceded by an operand then monadic minus - that is the following operand has its sign changed. If preceded by an operand and followed by an operand then subtract the following operand from the preceding one.
×	The operand preceding the operator is to be multiplied by the following operand.
/	The operand preceding the operator is to be divided by the following operand.
××	The operand preceding the operator is to be raised to the power of the operand following. (Note that the preceding operand cannot be negative if the operand following is not an integer).
//	The operand preceding the operator and the operand following are both, if nessesary converted to type INTEGER. The result of this division is then the integral part of the quotient. (A//B=SIGN(A/B)×ENTIER(ABS(A/B)))

Examples	Result
+ A	Do not change sign of A.
- B	Change the sign of B.
A + B	Add B to A.
A - B	Subtract B from A.
A * B	Multiply A by B.
A / B	Divide A by B.
A ** B	Raise A to the power B.
A // B	Change A and B to type INTEGER if of type REAL or REAL2. Divide A by B. The result is the integer part of A/B. Note If A or B are not of type INTEGER, a compilation warning is given since the ALGOL 60 report states that only INTEGER operands may be used.

b) Precedence of Arithmetic Operators

The precedence the arithmetic operators is:

1. **
2. *, /, //
3. +, -

This means that in a parenthesis-free expression, first all exponentiations will be carried out (from left to right), then all multiplications and divisions are executed (also from left to right), and finally all additions and subtractions are done. Parentheses may of course be inserted in the usual manner to give any desired grouping of subexpressions. (See also sec. 4.4)

Examples:

A ∗ B ∗ ∗ P	1. B and P are operands for ∗ ∗
	2. A and B ∗ ∗ P are operands for ∗
A + B/C ∗ D	1. B and C are operands for /
	2. B/C and D are operands for ∗
	3. A and B/C∗D are operands for +

c) Use of parentheses

It is suggested that parentheses by used as much as possible to group operations, so that the intended order of operations is immediately visible to the reader of a program.

4.2.5 Type of Arithmetic Expressions.

The value obtained by evaluating an arithmetic expression has a specific type according to the following rules.

a) Type of resulting value for operators +, -, ∗

Operand preceding is of type:	Operand following is of type			
	INTEGER	REAL	REAL2	COMPLEX
INTEGER	INTEGER	REAL	REAL2	COMPLEX
REAL	REAL	REAL	REAL2	COMPLEX
REAL2	REAL2	REAL2	REAL2	COMPLEX
COMPLEX	COMPLEX	COMPLEX	COMPLEX	COMPLEX

b) Type of resulting value for operators / and ∗∗

Operand preceding is of type	Operand following is of type			
	INTEGER	REAL	REAL2	COMPLEX
INTEGER	REAL	REAL	REAL2	COMPLEX
REAL	REAL	REAL	REAL2	COMPLEX
REAL2	REAL2	REAL2	REAL2	COMPLEX
COMPLEX	COMPLEX	COMPLEX	COMPLEX	COMPLEX

c) Type of resulting value for the operator //

is always INTEGER, if the types of the operand are INTEGER, REAL or REAL2.

If either of the operands are of any other type, a compik-time error will occur.

Example:

If the following declarations are used

INTEGER	I$
REAL	R$
REAL2	D$
COMPLEX	C$

then

Expression	has type
I * I	INTEGER
I / R	REAL
D + R	REAL2
C - D + I	COMPLEX
I ** I	REAL
D // R	INTEGER

4.3 Boolean Expressions

Meaning - A Boolean expression is a rule for Computing a Boolean value, that is, TRUE or FALSE.

Type of Value

A Boolean expression may only produce a value of type BOOLEAN.

Boolean Operands

Boolean Constants - are written as the character sequences TRUE or FALSE for the appropriate values.

Boolean Variables

Boolean variables are those variables whose identifiers have been declared to have type BOOLEAN. They may be simple or subscripted (that is, a member of a BOOLEAN array).

Boolean Type Procedures

The declaration of a type procedure is described in section 7.2.
In a Boolean expression, procedures of type BOOLEAN may occur.
The standard procedures which return a value of type BOOLEAN (for example ALPHABETIC and NUMERIC) may be used in Boolean expressions.

4.3.1 Boolean Operators.

a) The following Boolean operators are defined in NU ALGOL to have the following meanings only if A and B are BOOLEAN expressions.

Boolean Operators

Expression	Meaning	Value of expression			
		A=TRUE B=TRUE	A=TRUE B=FALSE	A=FALSE B=TRUE	A=FALSE B=FALSE
NOT A	(unary) negation	FALSE	FALSE	TRUE	TRUE
A OR B	disjunction	TRUE	TRUE	TRUE	FALSE
A AND B	conjunction	TRUE	FALSE	FALSE	FALSE
A IMPL B	implication	TRUE	FALSE	TRUE	TRUE
A EQIV B	equivalence	TRUE	FALSE	FALSE	TRUE
A XOR B	exclusive "or"	FALSE	TRUE	TRUE	FALSE

b) Precedence of Boolean Operators

1. NOT
2. AND
3. XOR, OR
4. IMPL
5. EQIV

The remarks on the precedence of the arithmetic operators apply also for Boolean operators (see sections 4.2.4 and 4.4).

4.3.2 Relational Operators.

a) The following relational operators are defined in NU ALGOL to have the following meaning. C and D are arithmetic or string expressions.

Note If D or C are of type COMPLEX or STRING only EQL or NEQ may be used.

Relational Operators

Expression	Meaning	Value of Expression		
		for C > D	for C = D	for C < D
C LSS D	LeSS than	FALSE	FALSE	TRUE
C LEQ D	Less than or EQual	FALSE	TRUE	TRUE
C EQL D	EQuaL	FALSE	TRUE	FALSE
C GEQ D	Greater than or EQual	TRUE	TRUE	FALSE
C GTR D	GreaTeR than	TRUE	FALSE	FALSE
C NEQ D	Not EQual	TRUE	FALSE	TRUE

b) For strings, the comparison to determine equality or non-equality will be made on a character by character basis, starting with the leftmost character. If the strings are of unequal length, the string of shorter length will be considered to be filled with blanks to the length of the longer.

Examples:

For the following declarations and statements

```
STRING          S(7)$
REAL            X,Y$
INTEGER ARRAY   IA(-5:2)$
BOOLEAN         B$
S = 'ABCDEFG'$  X = 12 4$     Y = 15 0$
IA(-5) = 22$    IA(0) = 21$   B = TRUE$
```

The expression	has the value
X GTR Y	FALSE
S EQL 'ABCDEF'	FALSE
S NEQ 'ACDEFGA'	TRUE
IA(-5) LSS IA(0)	FALSE
IA(0) LEQ IA(-5)	TRUE
NOT B	FALSE
Y GEQ X	TRUE
NOT B AND X GTR Y	FALSE
S EQL 'ABCDEFG' OR S EQL 'XYZ'	TRUE
IA(-5) LEQ 12 IMPL B	TRUE
Y GTR 10.0 EQIV X LSS 12.0	FALSE
NOT B XOR X EQL Y	FALSE

4.4 Precedence of Arithmetic, Boolean and Relational Operators.

1. **
2. * / //
3. + -
4. Relational operators LSS, LEQ, EQL, GEQ, GTR, NEQ
5. NOT
6. AND
7. OR, XOR
8. IMPL
9. EQIV

Operations are carried out in order of ascending rank number.

Operations of equal rank are carried out from left to right.

Parentheses may be used to change the order of operations. The use of parentheses is suggested to ensure that the calculation wanted is the one that is performed. (See also section 4.2.4).

Example:

```
     BOOLEAN      A, B, C, D $
     INTEGER      X, Y, Z, W, T $
A=A EQIV B IMPL C OR D AND NOT Y+Z*W**T GTR X $
```

Evaluation:

```
 1.  W**T
 2.  Z*(W**T)
 3.  Y+(Z*(W**T))
 4.  (Y+(Z*(W**T))) GTR X
 5.  NOT (Y+(Z*(W**T))) GTR X)
 6.  D AND (NOT((Y+(Z*(W**T))) GTR X))
 7.  C OR (D AND (NOT((Y+(Z*(W**T))) GTR X)))
 8.  B IMPL (result of 7)
 9.  A EQIV (result of 8)
10.  A =    (result of 9)
```

4.5 String Expressions.

Meaning

A string expression is a rule for obtaining a string of characters.

4.5.1 String Operands.

String Constants - are written as a string of characters not containing a string quote (') and enclosed by string quotes.

Examples:

```
'NU ALGOL'
'THIS IS A STRING CONSTANT'
'BAD * ? ! / + - WORDS'
```

String Variables

String variables are those variables whose identifiers have been declared to have type STRING.

String variables may be simple or subscripted, that is, a member of a STRING ARRAY.

4.5.2 String Operators.

For strings no operators giving a string result are defined.

a) Arithmetic Operations on Strings.

Arithmetic operators may be used between string operands if the string involved contain only digits in the form of INTEGER constants (including sign).

If the string is not in the form of an integer constant (either contains non-digits or too many digits) then a run-time error message will be given.

If the string is in the form of an integer constant then the value of this integer will be used as the operand.

Example:

```
STRING S(12) $      INTEGER X $
S = 'ANS IS 56345' $
X = S(8,5)+20 $
COMMENT THE VALUE ASSIGNED TO X IS 56365 $
```

b) Relational Operators.

The equality of strings may be tested using the relational operators EQL and NEQ. (See section 4.3.2).

4.5.3 Substrings.

Purpose

To refer to a part of a string variable, a substring may be used.

a) Declared Substring

Substrings may be declared in the declaration of the main string (See section 3.2.2).

b) Substring expressions.

A substring of a main string may be referenced by giving a start character number in the main string and the length of the substring on the form

<string identifier>(<start character number>,<length of substring>)

Example:

STRING   K(50)$

K(20,6) is a substring referring to characters 20, 21, 22 23, 24, 25 in the main string K.

If no length is given, the substring is assumed to consist of one character.

Example:

K(29) is a substring consisting of character number 29 in the main string K.

If no start position or length is given then the main string is referred to

Example:

STRING K(50)$
K and K(1,50) are equivalent

c) Substrings_of_members_of_string_arrays.

A reference to a substring of a subscripted string variable is written on the form

<string array identifier>(<start character number>, <length of substring>:<subscript, or subscripts separated by commas>).

Example:

STRING ARRAY SA(10:0:10,1:2)$ defines a string array consisting of 22 strings each of 10 characters.
SA(5,2:1,2) is the substring made up of characters 5 and 6 of the element SA(1,2).
SA(10:0,1) is the substring made of character 10 of the array element SA(0,1).

The declaration of substrings of string array variables is described in section 3.3.3

4.6 Designational Expressions.

ALGOL statements are executed one after another in the order they appear in the program, unless a GO TO statement forces the execution to begin at a different point in the program. This point is given by the value of a designational expression.

Form

A designational expression may be either

i) a label or
ii) a switch indentifier with an index or
iii) IF <Boolean expression> THEN <simple designational expression>
EXPRESSION
ELSE <designational expression>

where Boolean expression is described in section 4.3. Simple designational expression is either (i) or (ii) or (iii) enclosed in parentheses.

Meaning:

i) A label refers to that point in the program where the label is declared (see section 4.6.1).

ii) A switch idnetifier with an index (say i) refers to the designational expression in the i^{th} position of the list of designational expressions in the switch declaration (see section 4.6.2). If an actual switch index is less t than 1 or greater than the number of designational expressions in the list, then GOTO statement is not executed.

iii) In the case of the designational expression IF <Boolean expression> THEN <simple designational expression> ELSE <designational expression>, the simple designational expression is used if the Boolean expression evaluated to the value TRUE, the designational expression is used if the Boolean expression evaluated to the value FALSE.

4.6.1 Labels

Purpose

By the use of a GOTO statement, control may be transferred to a specific program point. This program point must then be given a name, called a label.

Label Declaration

Labels are declared by placing an identifier in front of a sattement and separating it from the statement by the colon symbol (:).

Example: LAB1 : X = 5$

Because in NU ALGOL a label is an identifier (see section 2.2), numeric labels are not allowed.

Only one label with the same identifier may be used within a block.

Labels are local to the block in which they have been declared.

4.6.2 Switches.

Purpose

A switch allows the programmer to select a certain label depending on an index.

The SWITCH declaration has the following form

SWITCH<identifier>=<list of designational expressions>$

where identifier is as defined in section 2.2. List of designational expressions is a set of designational expressions separated by commas. Designational expression is described below.

Examples:

```
SWITCH S1W2 = P1, IF A GTR 2 THEN L ELSE Z $
SWITCH S1W3 = S1W2(1), S1W2(2) $
COMMENT NOTICE THAT A SWITCH IDENTIFIER WITH INDEX IS A
DESIGNATIONAL EXPRESSION $
```

4.7 Conditional Expressions

Purpose - It is possible to use different operands in an expression according to the value of a Boolean expression by placing the operands in a conditional expression.

Form - The conditional expression has the form

IF <Boolean expression> THEN <simple expression>
ELSE <expression>>

where Boolean expression is described in section 4.3.
Simple expression is any of the expressions (aritmetic, Boolean or string) described in section 4, or a conditional expression enclosed in parentheses.
Expression can be either a simple expression as described above or a conditional expression.

Rules for expressions

a) The 'simple expression' and the 'expression' used in an expression must be of the same kind. That is both must be of kind arithmetic, boolean, string, or designational.

b) If the 'simple expression' and the 'expression' are both of kind arithmetic but are of different types, then the value of the expression will have the type given by the following table.

c) Conditional expressions used as operands must be enclosed by parentheses.

Resulting type of expression

Simple expression has type	Expression has type			
	INTEGER	REAL	REAL2	COMPLEX
INTEGER	INTEGER	REAL	REAL2	COMPLEX
REAL	REAL	REAL	REAL2	COMPLEX
REAL2	REAL2	REAL2	REAL2	COMPLEX
COMPLEX	COMPLEX	COMPLEX	COMPLEX	COMPLEX

Examples:

```
BOOLEAN     B$
REAL        X,Y$
REAL2       D,E$
COMPLEX     C$
STRING      LETTERS(14)$
```

$$\texttt{X = }\underbrace{\texttt{IF B THEN X ELSE D \$}}_{\text{Arithmetic expression of type REAL2}}$$

$$\texttt{LETTERS = }\underbrace{\texttt{IF X GTR Y THEN LETTERS (1,4) ELSE LETTERS (4,8)\$}}_{\text{String expression}}$$

$$\texttt{B = }\underbrace{\texttt{IF D LSS E THEN NOT B ELSE D LSS E\$}}_{\text{Boolean expression}}$$

$$\texttt{C = (IF B THEN (IF NOT B THEN X ELSE Y)}$$

$$\underbrace{\texttt{ELSE IF X GTR Y THEN D ELSE E)}}_{\text{Arithmetic expression of type REAL2}}\quad\texttt{+ 20\$}$$

5 STATEMENTS.

5.1 Assignment Statements.

$$V_1 = V_2 = \text{-------} = V_n = E\$$$

Where the V_i are variables (either simple or subscripted) and E is an expression. The sign (=) or (:=) means "becomes" or "gets the value of".

a) Rules for performing assignment

If V is a subscripted variable, evaluate its subscript expressions, thus determing the actual variable. If there is more than one V in the statement, determine the actual variables from left to right.

Evaluate the expression E and assign this value to the variable or variables determined by the rule above.

b) Type rule for multiple assignment statements

All variables in the left part list (V_i) - that is, all variables to the left of the rightmost replacement sign (=) must be of the same type.

Examples:

```
INTEGER ARRAY     A(1:5)$
REAL              X,Y$
REAL ARRAY        Z(3:10)$
INTEGER           I,J$
I = 5$  J = 4$    COMMENT SIMPLE ASSIGNMENT $
A(I) = I = I + J$ COMMENT A(5) GETS THE VALUE 9,
                          I    GETS THE VALUE 9$
X = Y = I$        COMMENT ONLY VARIABLES IN THE LEFT
                  PART LIST MUST BE OF SAME TYPE, HERE
                  X BECOMES 5.0, Y BECOMES 5.0$
```

c) Transfer functions in assignment statements

If the type of the expression is different from that of the variable or variables in the assignment statement, then automatic type transfer occurs, if possible, according to the following rules.

Transfer Functions

Type of Variable	Type of Expression					
	INTEGER	REAL	REAL2	COMPLEX	STRING	BOOLEAN
INTEGER		Rounded to INTEGER	Rounded to INTEGER	Not Allowed	Changed to INTEGER if possible	Not Allowed
REAL	Converted to REAL		Truncated to REAL	Not Allowed	Not Allowed	Not Allowed
REAL2	Converted to REAL2	Zero filled to REAL2		Not Allowed	Not Allowed	Not Allowed
COMPLEX	Becomes real part of COMPLEX	Becomes real part of COMPLEX	Truncated to real part of COMPLEX		Not Allowed	Not Allowed
STRING	Integer is left justified in string	Not Allowed	Not Allowed	Not Allowed	See below	Not Allowed
BOOLEAN	Not Allowed	Not Allowed	Not Allowed	Not Allowed	Not Allowed	

d) String Assignment

If the string expression has fewer characters than the string variable, the remainder of the string variable is filled with blanks.

If the string expression has more characters than the string variable then these extra characters are not transferred to the string variable.

The assignment is a character by character transfer starting from the leftmost character.

Note the following example

```
     STRING  ST(15) $
             ST = 'ABC' $
             ST(2,14) = ST(1,14) $
COMMENT THE RESULT OF THIS ASSIGNMENT IS THAT THE ENTIRE
STRING ST IS 'AAAAAAAAAAAAAA'.$
```

5.2 GO TO Statements.

Purpose

The purpose of a GO TO Statement is to break the normal sequence of execution of statements in a program.

The statement executed after a GO TO Statement is the statement following the label given by the designational expression in the GO TO Statement. (Labels and designational expressions are described in section 4.6).

Form

There are three possible ways of writing a GO TO statement. All have the same meaning.

```
GO TO<designational expression>$
GOTO <designational expression>$
GO   <designational expression>$
```

Examples:

```
     SWITCH  KF = XY,ZW $   BOOLEAN B $
     GO TO XY $
SW:  GOTO KF(1)$
     GO IF B THEN ZW ELSE XY $
XY:  GO TO IF NOT B THEN KF(2) ELSE SW $
```

5.3 Compound Statements.

Definition

A compound statement is a group of ALGOL statements enclosed by the words BEGIN and END

Action

A compound statement may be wherever one ALGOL statement is allowed.

Use

Compound statements are very useful in conditional and repetitive statements (see section 5.4 and 5.5) where only one statement is allowed.

Examples:

```
BOOLEAN B$      REAL X,Y,Z $
IF   B   THEN
BEGIN  X = 5.0$  Y = 15.0$  Z = 22.1$
END $
FOR X = 20.0 STEP 1 UNTIL 50.0 DO
BEGIN  Y = Y+ X $  Z = X * 20.0 + Z $
END $
```

5.4 Conditional Statements.

Purpose

Conditional statements may be used to select the next statement depending on the value of a Boolean expression.

Forms

There are two types of conditional statements - one with alternative and one without. The forms are given below.

a) Conditional statement WITHOUT alternative

IF<Boolean expression>THEN<unconditional statement>$
where Boolean expression is described in section 4.3.
An unconditional statement is either any statement other than a conditional statement, including a compound statement, or a conditional statement enclosed by BEGIN and END.

Example:

```
IF  A GTR B THEN A = A - B $
```

b) Conditional statement WITH alternative

IF<Boolean expression>THEN<unconditional statement>
ELSE<statement>$

where Boolean expression is described in section 4.3,
- unconditional statement is any statement other than a conditional statement, including a compound statement. Notice that a $ or; must never appear before ELSE.
- statement is any statement including a conditional statement or a compound statement.

Example:

```
IF A GTR B THEN A = A - B ELSE A = B - A $
```

Actions

a) Conditional statement WITHOUT alternative

Boolean expression evaluates to	Action
TRUE	Execute unconditional statement after THEN
FALSE	Execute statement after conditional statement

b) Conditional statement WITH alternative

Boolean expression evaluates to	Action
TRUE	Execute unconditional statement after THEN
FALSE	Execute statement after ELSE

Examples:

```
BEGIN
   REAL X,Y$      BOOLEAN B $
   SWITCH  SK = LAB,LIN $
      IF NOT B  THEN  X = Y = 20.1 $
   COMMENT B IS FALSE, SO X AND Y ARE SET TO 20.1 $
LIN: IF X NEQ Y THEN B = FALSE
                ELSE B = TRUE $
   COMMENT X AND Y ARE EQUAL, SO B IS SET TO TRUE $
   IF B THEN BEGIN IF X EQL 25.0 THEN Y = 24.9 END
             ELSE GO TO SK(2) $
   COMMENT B IS TRUE BUT X IS NOT EQUAL TO 25.0, SO
   THE NEXT STATEMENT IS EXECUTED $
      B = FALSE $
LAB: IF Y GTR 20.1 THEN GO TO LIN $
   COMMENT Y EQUALS 20.1, SO THE PROGRAM FINISHES $
END $
```

5.5 Repetition Statements - FOR Statements.

Purpose

The repetition statement allows a given statement to be executed several times.

Form

FOR V = <list of FOR list elements>DO<statement>$
where V must be a variable. This variable is called the controlled variable

- FOR list element is described below.
- statement is one ALGOL statement of any kind, including conditional or compound statements.

Rules for the controlled variable

The controlled variable may only be of type INTEGER or REAL. If the controlled variable is a formal paramter, then the type of the actual parameter must coincide with that of the formal. When the controlled variable is subscripted, the subscript(s) are evaluated once, before entering the loop.

FOR list elements

There are three kinds of FOR list elements.

a) An arithmetic expression

Form

The for list element is an arithmetic expression of type INTEGER or REAL only.
If the controlled variable is of type INTEGER when an expression is of type REAL, the value of the expression will be rounded to INTEGER.

Action

Step - (The step numbers are used in the example, as well as to illustrate the order).

1. Evaluate the expression.
2. Assign the value to the controlled variable, converting to the type of the controlled variable if necessary.
3. Execute the statement following DO.
4. If there are no more for list elements then execute the next statement.
5. If there is another for list element, repeat from step 1.

Example:

```
INTEGER A,B,C,TOTAL $
A = 10$   B = 5$
FOR C = A + 5, A + 20, B + 1, B DO
    TOTAL = TOTAL + C $
```

Action A has the value 10, B the value 5.

(coutd. on next page)

Step	Expression		Value of C	Value of TOTAL
	Number	Value	0	0
1	1	15		
2			15	
3				15
4	Another for list element follows			
5	2	30		
2			30	
3				45
4	Another for list element follows			
5	3	6		
2			6	
3				51
4	Another for list element follows			
5	4	5		
2			5	
3				56
4	No more for list elements go to next statement			

b) STEP UNTIL construction

Form

There are two forms for this for list element.

A STEP B UNTIL C

or

(A, B, C).

Notice that if the brackets are not present the latter is a group of FOR list elements.

In both cases A, B and C are all arithmetic expressions. They may only be of type INTEGER or REAL. If the controlled variable is of type INTEGER while any of the A, B or C are of type REAL, the value obtained is rounded to INTEGER. B is called the step. C is called the limit. A is called the initial value.

Action

1. Evaluate the expression A - call this value X.
2. Assign the value X to the controlled variable, converting it to the type of the controlled variable if necessary.
3. Evaluate the expressions B and C and convert to the type of the controlled variable if necessary.
4. If the value of B is negative then go to step 6.
5. If the value of X is greater than the value of C then go to step 10, otherwise go to step 7.
6. If the value of X is less than the value of C then go to step 10.
7. Execute the statement after DO.
8. Calculate the sum of the value of X and the value of B - call the result of this calculation X.
9. Start again at step 2.
10. If there are more FOR list elements start to perform them - (note that the controlled variable has been stepped) otherwise execute the statement after the FOR statement.

Examples:

1.
```
INTEGER I $  REAL J,K $
INTEGER ARRAY Z(1:4) $
   J = 10 4 $  K = 20 6 $  I = 2 $
FOR Z (I) = J + K STEP - J - I UNTIL - 41
   DO I = I +A (2) $
```

Action

In this example
the initial value expression A is J + K
the step B is J
the limit C is -41
the controlled variable is Z(2)

Step	Value of A	Value of B	Value of C	Value X	Value of Z(2)	Value of I	Value of J	Value of K
Start					0	2	10.4	20.6
1	30.0			30				
2					30			
3		-12	-41	-41				
4	Go to step 6							
6	30>-41 - do next step							
7						32		
8				18				
9	back to step 2							
2					18			
3		-42	-41					
4	Go to step 6							
6	18>-41 - do next step							
7						50		
8				-24				
9	Go to step 2							
2					-24			
3		-60	-41					
4	Go to step 6							
6	-24>-41 - do next step							
7						26		
8				-84				
9	Go to step 2							
2					-84			
3		-36	-41					
4	Go to step 6							
6	-84<-41 - Go to step 10							
10	No more FOR list elements, go to next statement							

2. In a more simple case set all members of an array to a value

```
REAL D $
REAL ARRAY DA(-25 : 20) $
INTEGER I $
FOR I = (-25,1,20) DO DA(I) = D $
```

3. Perform a group of statements N times.

```
INTEGER I,N $
FOR I = (1,1,N) DO
BEGIN
   READ (X) $   COMMENT WILL READ N CARDS $
   Y = 50 * X $
   WRITE (Y) $  COMMENT WILL PRINT N LINES $
END $
```

4. Set <u>specific</u> members of an array to a certain value

```
INTEGER I $   REAL ARRAY   X(1:200) $
REAL R $
FOR I = 1 STEP 1 UNTIL 5, 8, 9, 20 STEP 10
    UNTIL 60, 100, 200  DO
          X(I) = R $
COMMENT  X(1), X(2), X(3), X(4), X(5), X(8), X(9),
         X(20), X(30), X(40), X(50), X(60), X(100),
         X(200) WILL BE GIVEN THE VALUE OF R $
```

b) WHILE construction

Form

<Arithmetic expression>WHILE<Boolean expression>
where arithmetic and Boolean expressions are as described in section 4.

Action

1. Evaluate the arithmetic expression.
2. Assign the value of the arithmetic expression to the controlled variable, converting if necessary.
3. Evaluate the Boolean expression.
4. If the Boolean expression has the value FALSE then go to step 7.

5. Execute the statement after DO.
6. Go to step 1.
7. If there are no more FOR list elements, execute the statement after the FOR statement, otherwise take the next FOR list element.

<u>Examples</u>:

1.
```
INTEGER I, COUNT $
STRING  S(350), SD(21)$
SD = 'OVERWRITE BLANK AREAS' $
FOR I = I + 1 WHILE S(I) EQL ' ' AND I LSS 22 DO
        S(I) = SD(I) $
```

2. This FOR list element is useful when adding terms into a series
```
REAL  X,  TOTAL $
      X = 25.0 $
FOR   X = 0.5 × SQRT (X) WHILE X GTR 0.5 DO
      TOTAL = TOTAL +X $
```

Step	Value of Arithmetic Expression	Value of X	Value of Boolean Expression	Value of Total
Start		25.0		0.0
1	2.5			
2		2.5		
3			TRUE	
4	Value is TRUE, so do next step			
5				2.5
6	Go to step 1			
1	.791			
2		0.791		
3			TRUE	
4	Value is TRUE, so do next step			
5				3.291
6	Go to step 1			
1	.445			
2		.445		
3			FALSE	
4	Value is FALSE, so go to step 7			
7	No more FOR list elements, so do next statement			

d) <u>Special rules for FOR statements</u>

i) Upon exit from a FOR statement either because there are no more FOR list elements or because of a GO TO statement, the controlled variable has a specific value. This value may be calculated by referring to the rules for the type of FOR list element being used.

ii) A GOTO leading to a label withing the FOR statement is illegal. A label may however be used for a jump <u>within</u> the statement following DO.

5.6 <u>Other Types of Statements</u>.

<u>Input/Output Statements</u> are described in section 8.

<u>Procedure Statements</u> or calls on procedures which do not have a type are described in section 7.

<u>Blocks</u> as statements - are described in section 6.

The OPTION feature which may be used like a statement is described in section 9.

6 BLOCKS.

Purpose

The ALGOL block effects a grouping of a set of variables and the statements involving these variables. The block structure of ALGOL reflects the dynamic storage of variables, and may be used to economize on storage space. An ALGOL program is an example of a block.

Form

A block has the following form

```
BEGIN
<declarations>$    Block head
<statements>       Block body
END $
```

Notice that the only difference between a block and a compound statement is that a block has declarations.

6.1 Nested Blocks.

A block may appear in the body of another block. This inner block is then said to be nested in the outer block.

Example:

```
OUTERBL:  BEGIN
      REAL  A, B $
        A = 1.5 $  B = 2.6 $
INNERBL1: BEGIN
            INTEGER C, D $
            C = A + B $  D = A - B $
          END $
            A = 50.0 $
INNERBL2: BEGIN
            REAL E, F $
            E = A × B $      F = A/B $
          END $
            A = A + B $
END $
```

Here the blocks with the labels INNERBL1 and INNERBL2 are nested in the outer block with the label OUTERBL. The blocks with the labels INNERBL1 and INNERBL2 are non-nested.

6.2 Local and Global Identifiers.

Consider the following example, where the blocks B2 and B3 are nested in block B1.

```
BEGIN
    BEGIN
              B2
    END $
                          B1
    BEGIN
              B3
    END $
END $
```

a) Identifiers that are declared in B1 but not in B2 or B3, are local in B1 and global in B2 and B3.

b) Identifiers that are declared in B2 are undefined in B1 and B3. They are local in B2.

c) Identifiers declared in B3 are undefined in B1 and B3. They are local in B3.

d) If the same identifier is declared in both B1 and B2, then the declaration in B1 is ignored within B2. If the identifier is used in B1 or B3, the declaration given in B1 will be used.

e) Upon entering a blocks, variables are initialized to 0 if arithmetic, to FALSE if Boolean, and to blanks if string.

Examples:

1. In the previous example
   B1 is the block with the label OUTERBL,
   B2 is the block with the label INNERBL1,
   B3 is the block with the label INNERBL2.

Identifiers A and B are local to block OUTERBL, and global to blocks INNERBL1 and INNERBL2.
Identifiers C and D are local to block INNERBL1 and undefined in the other two blocks.
Identifiers E and F are local to block INNERBL2 and undefined in the other two blocks.

2.
```
BEGIN
   REAL A $
       A = 50.0 $    COMMENT HEREA IS LOCAL AND REAL $
   BEGIN
     INTEGER A $
       A = 5 $       COMMENT HEREA IS LOCAL AND INTEGER $
   END $
   BEGIN
       A = 25.0 $    COMMENT HEREA IS GLOBAL AND REAL $
   END $
 END $
```

6.3 Local and Global Labels.

Labels are declared, as explained in section 4.6.2, by placing an identifier and a : in front of the statement to which the label applies. Labels can thus be local or global depending on where they are declared.

Only labels which are local or global may be used in a designational expression in a certain block. That is, GO TO statements may only lead to statements in the same block or in an enclosing block, never to statements in a non-nested block.

Note - in NU ALGOL, the outermost block may not have a label, since jumps to this label have no meaning.

6.4 Use of Blocks.

a) To give the values to expressions in declarations

In section 3 - Declarations - it is stated that the bounds for arrays, and the length of a string may be arithmetic expressions. Variables or type procedures may be used in these expressions only if they are global to the block in which the declaration appears.

b) To save core storage

Non-nested blocks on the same block level use the same area of core for the storage of their local variables.

Examples:

```
BEGIN
      INTEGER X,Y,Z,N $
      READ (X,Y,Z,N) $
    BEGIN
      REAL ARRAY A(1:X,1:Y), B(1:Y,1:Z) $
      STRING ST(X+Y+Z-N) $
    END $
    BEGIN
      INTEGER ARRAY K(N:X,N:Z) $
      COMMENT THIS ARRAY USES THE SAME CORE
      AREA AS A AND B IN THE BLOCK ABOVE $
    END $
END $
```

7 PROCEDURES AND TYPE PROCEDURES.

7.1 Procedures.

7.1.1 Purpose.

When a group of statements are used in several places in a program, possibly with different values of the variables, then this coding may be written once in a procedure declaration and used whenever necessary through the means of procedure calls or procedure statements.

7.1.2 The Procedure Declaration.

Form

Procedure head	PROCEDURE identifier (formal parameter list) $ VALUE<identifier list>$ <specifications>$
Procedure body	<statement>$

where identifier is as described in section 2.2.
- formal parameter is described below.
- specification is described below.

7.1.3 Identifiers in the Procedure Body.

Local

The statement which is the procedure body may be a block. Identifiers declared in the block are local to the block. (See section 6.2).

Global

Identifiers declared in the block containing the procedure declaration or enclosing blocks are global to the procedure body and may be used by the statement.

Example:

```
BEGIN $
            INTEGER   I $
            PROCEDURE P $   COMMENT PROCEDURE HEAD WITH
                            NO PARAMETERS OR SPECIFICATIONS $
      BEGIN
            INTEGER   K $   COMMENT K IS LOCAL $
              K = 5 $
              I = I + K $   COMMENT I IS GLOBAL $
      END $
END $
```

The selection of the actual variables to be used in the statement is done when execution of the procedure is involved. However, it is necessary to have representative variables in the procedure declaration to allow the construction of a correct statement. These representative variables are called formal parameters. The variables used by the procedure during execution are called the actual parameters.

7.1.4 Specifications.

Purpose

The specifications give the type and kind of the formal parameters and may also indicate the modes of transmission of the actual parameters.

Form

The form of a specification is

<specifier><list of identifiers>$

where the list of identifiers has the usual meaning, except that in this case the identifiers may only be formal parameters.
The following table gives the possible specifiers.

Use the specifier	When a formal parameter is to be
INTEGER REAL REAL2 COMPLEX BOOLEAN STRING	A simple variable of the specified type
INTEGER ARRAY REAL ARRAY or ARRAY REAL2 ARRAY COMPLEX ARRAY BOOLEAN ARRAY STRING ARRAY	An array of the specified type
LABEL	A label
SWITCH	A switch
PROCEDURE	A procedure
INTEGER PROCEDURE REAL PROCEDURE REAL2 PROCEDURE BOOLEAN PROCEDURE COMPLEX PROCEDURE	A type procedure of the specified type
FORMAT	A format
LIST	A list
VALUE	Special meaning see section 7.1.7.

Note: The VALUE Specification must come before all other specifications.

7.1.5 The Procedure Body.

The procedure body must be only one statement. This statement may be a compound statement or a blcok.

A formal parameter used on the left hand side of an assignment statement must have a variable for actual parameter.

Example of procedure declaration:

```
PROCEDURE EXAMPLE (A,B,ANS,C)$
  VALUE B $                  COMMENT VALUE SPECIFICATION $
     REAL ARRAY B $          COMMENT OTHER SPECIFICATIONS $
     INTEGER A $
     REAL ANS $
     LABEL C $
  BEGIN                      COMMENT START OF PROCEDURE BODY $
     REAL2 TEMP $            COMMENT LOCAL VARIABLE $
     TEMP = B(A) + B(A+1) $
     ANS = TEMP/2.0&&4 $
     IF ANS LSS 0.0 THEN GO TO C $
  END $
```

7.1.6 Classification of Formal Parameters.

The formal parameters may be classified by the way they are used in the procedure body.

Arguments - are those parameters (variables or type procedures) which bring into the procedure values that will be used by the procedure body.

Results - are those parameters which are assigned values in the procedure body.

Exits - consist of those formal parameters which are labels or switches. Exits may be used as a special way of returning from a procedure.

Note: A parameter may be both an argument and a result.

7.1.7 VALUE Specification.

(The main implications of this specification can be seen in section 7.1.11 - Execution of a procedure statement).

However, the following kinds of formal parameters may not be placed in a VALUE specification.
LABEL, SWITCH, FORMAT, PROCEDURE, LIST

The VALUE specification causes the value or values of the formal parameter to be copied into a termporary area. These values can then be manipulated or changed without destroying the values of the actual parameter.

A main advantage of the VALUE specification is that if the actual parameters are expressions they are evaluated only once.

Example:

```
PROCEDURE  COUNT (N,ANS) $
VALUE N $      COMMENT N IS AN ARGUMENT WHICH SHOULD
               NOT BE CHANGED $
INTEGER N, ANS $  COMMENT ANS IS THE RESULT $
BEGIN
   INTEGER I,J $
   FOR J = N/2 WHILE N NEQ 0 DO
   BEGIN
      IF 2*J NEQ N THEN I = I + 1 $
      N = N//2 $  COMMENT NOTICE THAT THE FORMAL PARAMETER
                  IS CHANGED, BUT NOT THE ACTUAL $
   END $
   ANS = I $
END $
```

7.1.8 Comments in a Procedure Head.

COMMENTS may be placed anywhere in the procedure declaration after the delimiter $ or ; (see section 9). Comments may also be placed in the formal parameter list by using the following delimiter instead of a comma.

)string of letters not including : or $ followed by :(

Examples:

```
1. PROCEDURE EXAMPLE (A,N,S) $
   COMMENT N IS THE DIMENSION OF THE ARRAY A
   S IS AN EXIT $

2. PROCEDURE EXAMPLE (A) IS AN ARRAY WITH DIMENSION : (N)
   IF ERROR EXIT TO : (S) $
   COMMENT THE FORMAL PARAMETERS ARE A,N,S $
```

7.1.9 The Procedure Statement.

Purpose

A procedure statement "calls" a declared procedure and transmits actual parameters corresponding to the formals of the procedure. A call to a procedure will effect the execution of the procedure body.

Form

<identifier>(<actual parameter list>) $
where identifier is the identifier of the wanted procedure.
- actual parameter list is a list of variables or expressions.

7.1.10 The Actual Parameter List.

The i'th element of the actual parameter list corresponds to the i'th parameter in the formal parameter list.

There must be the same number of actual parameters as there are formal parameters for a certain procedure.

For type and kind correspondence of actual and formal parameters, the following rules apply:

Formal parameter	Actual parameter can be
Simple variable	Simple or subscripted variable, constant, or expression of the same type as the formal parameter or of a type that can be converted to that of the formal parameter. (See restriction below).
Array	Array of the same type and with the same number of subscripts as the array used in the procedure body.
Label	Designational expression
Switch	Switch
Procedure	Procedure with a formal parameter list compatible with the list of actual parameters used in the call of the formal procedure.
Type procedure	Type procedure of the same type as the formal procedure or of a type compatible to that of the formal procedure and with a formal parameter list compatible with the actual parameter list used in the call of the formal procedure.

Restriction

A formal parameter used on the left side of an assignment statement or as the controlled variable in a FOR statement can only have as actual parameter a simple or subscripted variable, not

an expression or a constant.
Notice that a formal parameter whose actual parameter is a constant or an expression may be used for temporary storage if the formal parameter is VALUE specified. In this case, once something has been assigned to the formal parameter, the value of the actual parameter is lost to further calculations in the procedure.

Examples of procedure statements:

1. For the procedure declared in section 7.1.5.
```
REAL ARRAY ARY(1:25) $  INTEGER RESULT $
EXAMPLE (15,ARY,RESULT,L1) $
L1:
```
2. For the procedure declared in Section 7.1.7.
```
INTEGER K,SIZE $
K = 25 $  COUNT (K,SIZE) $
```

7.1.11 Execution of a Procedure Statement.

The procedure statement causes the execution of the statement in the procedure body just as if the procedure statement were replaced by the statement in the procedure body with the following modifications.

a) All formal parameters which have not been VALUE specified (name parameters), are treated as if they were textually replaced by the corresponding actual parameters in the procedure body.

b) Formal parameters which have been VALUE specified are the evaluated, and these values are assigned to the formal parameters, which are then used in the procedure body.

Examples:

1. Without value specification

```
COMMENT PROCEDURE DECLARATION $
PROCEDURE VOLUME (LENGTH,WIDTH,HEIGHT,ANS) $
REAL LENGTH,WIDTH,HEIGHT,ANS $
   ANS = LENGTH × WIDTH × HEIGHT $
COMMENT PROCEDURE STATEMENT $
   VOLUME (P+5.0,Q+3.1,Z+4.0, RESULT) $
```

The procedure statement is executed as if the following statement had been written,

```
RESULT = (P+5.0) × (Q+3.1) × (Z+4.0) $
```

2. With value specification

```
PROCEDURE VOLUME (LENGTH,WIDTH,HEIGHT,ANS) $
VALUE LENGTH,WIDTH,HEIGHT $
   REAL LENGTH,WIDTH,HEIGHT,ANS $
   ANS = LENGTH × WIDTH × HEIGHT $
COMMENT PROCEDURE STATEMENT $
   VOLUME (P+5.0,Q+3.1,Z+4.0, RESULT) $
```

The procedure statement is executed as if the following block had been written in its place.

```
BEGIN
   REAL LENGTH,WIDTH,HEIGHT $
      LENGTH = P+5.0 $
      WIDTH  = Q+3.1 $
      HEIGHT = Z+4.0 $
      RESULT = LENGTH × WIDTH × HEIGHT $
   COMMENT NOTE THAT THE ACTUAL PARAMETER RESULT IS STILL
   USED BECAUSE ANS WAS NOT IN THE VALUE SPECIFICATION $
END $
```

7.1.12 Recursivity.

A procedure may be called within its own procedure declaration. This feature is known as the recursive use of a procedure and is fully implemented in NU ALGOL.

7.2 Type Procedures.

7.2.1 Introduction.

Procedures will often calculate a single value.
Type procedures calculate a value and assign this value to the identifier given as the name of the procedure.
All of the rules for procedures stated in section 7.1 apply with a few added rules.

7.2.2 The Type Procedure Declaration.

<type> PROCEDURE<identifier>(<formal parameter list>) $
VALUE<identifier list> $
<specifications>$
<statements> $

where - <type> is described in section 3.2.
- identifier is described in section 2.2.
- formal parameter list, VALUE specifications are described in sections 7.1.

The statement should contain an assignment statement which assigns a value to the identifier used as the name of the procedure.

7.2.3 Use of a Type Procedure.

A type procedure may be used as an operand in an expression by using the following construction

<identifier>(<actual parameter list>)

(See also section 4 concerning operands in expressions).

In its declaration, the type procedure identifier may be used in an expression. This use is recursive because the procedure uses itself in the calculation. (See 7.1.11).

The standard procedures (library functions) are examples of type procedures. However, the standard procedures do not have to be declared.

Examples:

```
1. COMMENT TYPE PROCEDURE DECLARATION $
   REAL PROCEDURE VOLUME (LENGTH,WIDTH,HEIGHT) $
   VALUE LENGTH,WIDTH,HEIGHT $
   REAL LENGTH,WIDTH,HEIGHT $
   VOLUME = LENGTH * WIDTH * HEIGHT $
   COMMENT USE OF A TYPE PROCEDURE $
      P= 5.0 $  Q = 3.0 $  Z = 4.0 $
      WRITE (VOLUME (P+5.0,Q+3.1,Z+4.0)) $
```

This statement is executed as if the following block had been written:

```
BEGIN
   REAL LENGTH,WIDTH,HEIGHT,VOLUME $
   LENGTH = P+5.0 $
   WIDTH = Q+3.1 $
   HEIGHT = Z+4.0 $
   VOLUME = LENGTH × WIDTH × HEIGHT $
   WRITE (VOLUME) $
END $
```

7.3 External Procedures.

7.3.1 Introduction.

Use of External procedures.

External procedures allow the user to build a library of procedures which are useful to him and which can be easily accessed by declaring the required procedure to be EXTERNAL PROCEDURE.

Definition.

External procedures are procedures whose bodies do not appear in the main program. They are compiled separately and linked to the main program at its execution.

7.3.2 External Declaration.

Use

The external declaration informs the compiler of the existence of external procedures, of their type (if any), and of the proper manner to construct the necessary linkages.

Form

EXTERNAL <kind><type> PROCEDURE <identifier list> $

<type> is as defined in section 3.2.
If no type is given, then the external procedure is a pure procedure as described in section 7.1.
<kind> can be empty , ALGOL, FORTRAN, SLEUTH, or LIBRARY.

<empty> or ALGOL	means an external procedure in the ALGOL language. These are treated just like ordinary procedures declared within the program.
FORTRAN	means an external procedure written in the FORTRAN language.

SLEUTH and LIBRARY — means that the external procedure is written in the machine language SLEUTH II.

The following descriptions require an adequate knowledge of the EXEC II monitor system, FORTRAN and SLEUTH II.

7.3.3 ALGOL External Procedures.

Form

An ALGOL procedure declaration (see section 3) may be compiled separately if an E option (see section 9.2) is used on the ALGOL processor card.

Several procedures may be compiled using the same ALGOL processor card. A program containing externally compiled procedures does not require an enclosing BEGIN-END pair.

An ALGOL procedure compiled in this way will have only the first six characters of the procedure name marked as an entry point in the PCF.

Such a procedure may be referenced from another ALGOL program as an external procedure if the appropriate declaration and identifier is used.

Examples:

1. The externally compiled procedure.

```
∇E     ALG     <name>
    PROCEDURE RESIDUES (X,Y)$
    VALUE X,Y; REAL X,Y;
    BEGIN
     :
    END$
```

The main program

```
∇      ALG     <main name>
    BEGIN
    EXTERNAL PROCEDURE RESIDUES$
    REAL A,B$
     :
    RESIDUES (A,B)$
     :
END$
```

2. The externally compiled procedure.

```
∇E      ALG      <name>
    REAL PROCEDURE DET(A,N)$
    VALUE A,N$
    REAL ARRAY A$
    INTEGER N$
    BEGIN
    COMMENT THIS PROCEDURE FINDS THE DETERMINANT OF A
    REAL NxN MATRIX A, LEAVING A UNCHANGED AND ASSIGNING
    THE VALUE TO DET$
     .
     .
    DET:=---$
    END  DET$
```

The main program

```
∇       ALG      <main name>
    BEGIN
    REAL ARRAY MATRIX (1:10,1:10)$
    EXTERNAL REAL PROCEDURE DET$
     .
     .
     .
    WRITE(DET(MATRIX,10))$
     .
     .
    END OF MAIN PROGRAM$
```

7.3.4 FORTRAN Subprograms.

A FORTRAN SUBROUTINE or a FORTRAN FUNCTION may be made available to an ALGOL program by the declaration

EXTERNAL FORTRAN <type> PROCEDURE<identifier list>

where type is described in section 3.2 and identifier list in section 2.2.

Allowed parameters

Actual parameters in calls on such FORTRAN subprograms may be either expressions, arrays or labels. Procedures, string arrays, formats and lists may not be used. Strings may be used if the FORTRAN program handles them correctly.

The address of the string itself, not of the string descriptor, is transmitted. Labels may be used only if they are local to the block where the calls occurs.

Differences between FORTRAN function and subroutine

The inclusion of <type> in the declaration implies that the FORTRAN subprogram begins with <type> FUNCTION <name>. The absence of <type> implies that the FORTRAN subprogram begins with SUBROUTINE <name>.

Example:

```
   FORTRAN subprogram
   ∇      FOR   <namel>
          FUNCTION DET (A,N)
          DIMENSION A (N,N)
   C      DET FINDS THE DETERMINANT
   C      OF A REAL NxN MATRIX A,
   C      DESTROYING A (SINCE 'VALUE' IS
   C      NOT ALLOWED IN FORTRAN), AND
   C      ASSIGNING THE VALUE TO DET
          .
          .
          DET=---
          END
ALGOL mainprogram
∇         ALG   <name2>
          BEGIN
          ARRAY MATRIX (1:10,1:10)$
          EXTERNAL FORTRAN REAL PROCEDURE DET$
          .
          .
          WRITE (DET(MATRIX,10))$
          END OF MAIN PROGRAM$
```

7.3.5 Machine Language Procedures

Use

For certain special applications (for example, bit manipulation), machine language procedures are necessary. These available the use of the EXTERNAL SLEUTH or the EXTERNAL LIBRARY declarations.

Recursive and non-recursive

The following remarks apply only to non-recursive machine language procedures. The required information for writing recursive machine language procedures may be found in the ALGOL technical documentation.

Some rules for external machine language procedures

If <type> is used in the EXTERNAL procedure declaration, the value of the procedure must be left in register A0 for single word length types (BOOLEAN, INTEGER, REAL) and A0 and A1 for double word length types (COMPLEX, REAL2).

Only the volatile registers (B11,A0,A1,A2,A3,A4,A5,R1,R2,R3) may be used without restoring.

The first six characters of the name in the identifier list of the EXTERNAL PROCEDURE declaration must be the first six characters of the external entry point of the machine language procedure.

Simple strings and all arrays including string arrays used as parameters require special handling as explained in the next sections.

Comparison of the SLEUTH and LIBRARY procedures

	SLEUTH	LIBRARY
1. Method of parameter transmission	By means of parameter descriptors in core	Parameter addresses or values are delivered through the arithmetic registers.
2. Security	Checking of the legality of the actual parameter list must be done at run-time in the SLUTH procedure.	Full checking is done at compiletime.

	SLEUTH	LIBRARY
3. Speed of parameter transmission	Fairly slow because of the need for indirect addressing and run-time checking.	Fast because values of correct type and kind are delivered through registers.
4. Flexibility	Complete information available at run-time about the parameters.	Less flexible because allowable actual parameters are determined at compile-time.
5. Example		
Declaration:	EXTERNAL SLEUTH PROCEDURE ES$	EXTERNAL LIBRARY PROCEDURE EL(X,Y)$ REAL X,Y$$
Call:	ES (A,B)$ A and B may be of any type or kind.	EL(A,B)$ A and B must be REAL

7.3.5.1 The External SLEUTH Procedure

Declaration

EXTERNAL SLEUTH <type> PROCEDURE <identifier list> $

Examples:

```
EXTERNAL SLEUTH PROCEDURE  BIT, PACK $
EXTERNAL SLEUTH COMPLEX PROCEDURE ARRAYSUM$
```

Transmission of parameters

The call to a procedure which has been declared as an EXTERNAL SLEUTH PROCEDURE produces the following coding.

```
F5  FORM  30,6
F1  FORM  6,6,6,18
    LMJ   B11,<procedure name>
    F5    <not used> , <number of parameters>
    F1    <type>,<kind>,<base register>,<relative data address>
    ____
    ____
    ____
```

F1 is the parameter descriptor. There is one for each parameter in the call.

<type> can have the following values and meanings

1 INTEGER
2 REAL
3 BOOLEAN
4 COMPLEX
5 REAL2
7 STRING

<kind> can have the following values and meanings

1 Simple, constant, expression or subscripted variable
5 ARRAY
9 LABEL

The absolute data address (ADA) or location of the parameter is found from

<absolute data address>=<relative data address> + contents of <base register>

The <base register> field may be zero in which case nothing should be added to the data address.
Note that for all simple expressions the <absolute data address> contains the value of the parameter. For strings it contains the <string descriptor>. For arrays it contains the first word of the <array descriptor>.
Formal parameters may not be used as actual parameters to the call of a SLEUTH procedure.

Return point

The return point for a call with N parameters is the contents of register B11 + N + 1.

Example:

```
Call:   BIT (X,Y,Z,D,E,F)$
Return:  J  7,B11
```

Referencing of parameters

Values of parameters should be obtained by the use of an indirect command.

Example:

```
Call:                    PACK(A,B,C)$
To load value of B:  L   A2,*2,B11
If C is a label exit to C is J   *3,B11
```

See sec. 7.3.5.3, 7.3.5.4 and 7.3.5.5 for description of STRING, ARRAY and STRING ARRAY parameters respectively.

Program examples

Machine Language Program:

```
∇  ASM     <namel>
 . THE FOLLOWING PROGRAM HAS NO PURPOSE
 . OTHER THAN TO ILLUSTRATE THE ABOVE NOTES
$(1)                  EQUIV           SET UP MNEMONICS
EPS*                . HAS THE CALL EPS(INT,STRING,EXIT LABEL)$
             L,T1     A1,1,B11.       PICK UP TYPE AND KIND
             TE,U     A1,0101.        IF NOT SIMPLE
             J        *3,B11.         INTEGER GO TO ERROR EXIT
             L        A0,*1,B11.      PICK UP VALUE OF INTEGER
             TG,U     A0,1024.        IF THE INTEGER GEQ 1024
             J        *3,B11.         THEN GO TO ERROR EXIT
             L,T1     A1,2,B11.       PICK UP TYPE/KIND FOR
                                      SECOND PARAMETER
             TE,U     A1,0701.        IF NOT SIMPLE STRING
             J        *3,B11.         THEN GO TO ERROR EXIT
             L,H2     A1,*2,B11.      PICK UP ADDRESS FROM STRING
                                      DESCRIPTOR
             L        A5,1,A1.        PICK UP SECOND WORD OF STRING
             J        4,B11.          RETURN WITH A0 CONTAINING
                                      THE ACCEPTABLE INTEGER

 . THE NEXT ROUTINE
 . HAS THE CALL TIMER (ARRAY IDENTIFIER, ROW, COLUMN, ANSWER)
 . THIS ROUTINE MULTIPLIES THE FIRST THIRD
 . OF THE SPECIFIED ARRAY ELEMENT BY 3600
```

```
. THE SECOND THIRD BY 60 AND ADDS THE
. RESULTS TO THE THIRD THIRD

TIMER*   L        A0,*1,B11.   GIVES ADA
         L        A3,*3,B11.   PICK UP COLUMN
         MSI,H1   A3,1,A0.     MULTIPLY BY D2
         A        A3,*2,B11.   ADD ON ROW
         A,H1     A3,0,A0.     ADD ON BA
         L,H2     A1,0,A0.     PICK UP FA
         AU,H1    A1,0,A1.     ADD LENGTH TO FA
         TW       A1,A3.       IF ELEMENT NOT IN ARRAY
         J        MERR$.       GO TO SYSTEM ERROR EXIT
         L,T1     A0,0,A3.     PICK UP FIRST THIRD
         MSI,U    A0,60.       MULTIPLY BY 60
         A,T2     A0,0,A3.     ADD ON SECOND THIRD
         MSI,U    A0,60.       MULTIPLY BY 60
         A,T3     A0,0,A3.     ADD ON THIRD THIRD
         S        A0,*4,B11.   STORE RESULT IN
         J        5,B11.       FOURTH PARAMETER AND RETURN
         END.
```

<ins>Main program</ins>:

```
∇ ALG    <name2>
BEGIN
  EXTERNAL SLEUTH INTEGER PROCEDURE ESP$
  EXTERNAL SLEUTH PROCEDURE TIMER$
  INTEGER INT$
  STRING SOUT(4,SIN(7))$
  INTEGER ARRAY A1(1:50,0:10),RESULTS(-5:12)$
  WRITE(ESP(INT,SIN,ERR))$  GO TO L1$
  ERR: WRITE ('WRONG PARAMETER')$
  L1: TIMER(A1,5,9,RESULTS(12))$
END$
```

7.3.5.2 The External LIBRARY Procedure

Declaration

In order to make possible the compile-time checking of the parameters, the declaration of a LIBRARY procedure must contain specifications. The specification list is terminated by ; or $. The LIBRARY procedure therefore has the appearance of an ALGOL procedure with an empty body.

The form of the declaration is:

```
EXTERNAL LIBRARY<type>PROCEDURE<identifier>(<formal para-
                                                meter list>)$
<value part>
<specification part>$
```

Example:

```
EXTERNAL LIBRARY INTEGER PROCEDURE  COM(I,B1,CA)$
    VALUE I,B1$
    INTEGER I$
    BOOLEAN B1$
    COMPLEX ARRAY CA$$
```

Call

When a library procedure is called, parameter values or addresses are loaded into consecutive arithmetic registers. If the formal parameter is by value, the value of the actual parameter is loaded, otherwise the address of the parameter is loaded. The first parameter goes into A0, the second into A1 and so on. REAL2 or COMPLEX parameters called by value, occupy two consecutive registers. The number of parameters allowed in the call is therefore limited by the number of arithmetic registers available and can at most be 16.

Generally the type and kind of the formal and actual parameter must be the same. However, if the formal is a simple value parameter, the actual parameter need only be convertible to the formal type. A label must be local to the block where the call occurs.

The table below shows possible combinations of formal and actual parameters and the corresponding content of the arithmetic register. Blank fields indicate illegal combinations which will give compile-time errors.

Actual / Formal	simple or formal value simple	formal name simple	constant	subscr. variable	ex-pression	string formal and non-formal	array formal and non-formal	local label
value simple	value of parameter	value of parameter	value of constant	value of parameter	value of expression			
simple not by value	address of parameter			address of parameter				
value string						The string descriptor. Sec. 7.3.5.3		
string not by value						address of the string descriptor. Sec. 7.3.5.3		
array							address of the array descriptor. Sec. 7.3.5.4	
label								program address

Retur_point

Return from a LIBRARY procedure is always to 0,B11.

Example:

```
∇ ALG MAIN
BEGIN
     COMMENT THIS EXAMPLE SHOWS HOW TO PACK THREE INTEGER NUMBERS
     INTO ONE 1107/1108 COMPUTER WORD IN ORDER TO SAVE CORE SPACE,
     AND THEN UNPACKING THEM AGAIN FOR COMPUTATION. FOR SUCH
     PACKING THE NUMBERS MUST HAVE ABSOLUTE VALUES LESS THAN 2047.
     LARGER NUMBERS WILL BE TRUNCATED;
     INTEGER I,J,K,M,N;
     INTEGER ARRAY NUMBERS (1:10000);
     EXTERNAL LIBRARY PROCEDURE PACK (N,I,J,K);
        VALUE  I,J,K;
        INTEGER  N,I,J,K;
        COMMENT THE PROCEDURE PACKS I,J,K INTO N; ;

     EXTERNAL LIBRARY PROCEDURE UNPACK (N,I,J,K);
        INTEGER  N,I,J,K;
        COMMENT THE PROCEDURE UNPACKS N INTO I,J,K;;
     COMMENT READ 30000  NUMBERS FROM CARDS;

     FOR M = (1,1,10000) DO
     BEGIN
        READ(I,J,K);  PACK(NUMBERS(M),I,J,K);
        COMMENT THE CALL ON PACK WILL GENERATE THE FOLLOWING
        SEQUENCE:
                    L      A0,<address of array element>
                    L      A1,I,B2
                    L      A2,J,B2
                    L      A3,K,B2
                    LMJ    B11,PACK                ;
     END;
     COMMENT DO SOME CALCULATIONS;
     FOR M=(1,1,5000) DO
     BEGIN
        UNPACK(NUMBERS(M),I,J,K);
        COMMENT THE CALL ON UNPACK WILL GENERATE:
                    L      A0,<address of array element>
                    L,U    A1,I,B2
                    L,U    A2,J,B2
```

```
                L,U    A3,K,B2
                LMJ    B11,UNPACK                       ;
         N = I + J x K;
         UNPACK(NUMBERS(10000-M),I,J,K);
         N = N x K // I + J;
         WRITE(N);
      END;
END MAIN PROGRAM;

∇ ASM PUNP
        EQUIV.
PACKx.
        S,T1    A1,0,A0.    I GOES INTO T1
        S,T2    A2,0,A0.    J GOES INTO T2
        S,T3    A3,0,A0.    K GOES INTO T3
        J       0,B11.      RETURN TO MAIN PROGRAM

UNPACKx.
        L,T1    A4,0,A0.    GET NUMBER IN T1
        S       A4,0,A1.    STORE INTO I
        L,T2    A4,0,A0.
        S       A4,0,A2.
        L,T3    A4,0,A0.
        S       A4,0,A3.
        J       0,B11.      RETURN TO MAIN PROGRAM
        END.
```

7.3.5.3 String Parameters

The absolute data address is the location of the string descriptor . The string descriptor can be described as follows

```
        F4      FORM      12,6,18
                F4        <length>,<start>,<address>
```

where <length> is the number of characters in the string.

<start> is the start position of the string in the first word used S1=0, S2=1, S3=2, S4=3, S5=4, S6=5
It will be different from zero only for substrings.

<address> is the location of the first word used for the string.

7.3.5.4 Array Parameters

The absolute data address (ADA) is the start address of the array descriptor.

The array descriptor has the following format.

Address	H1	H2	
ADA	BA	FA	
ADA+1	D2	D3	Dope vector elements -
ADA+2	D4	D5	as many as required
ADA+3	D6	D7	maximum of 9 since the
ADA+4	D8	D9	maximum number of dimen-
ADA+5	D10		sions is 10.

BA - Base Address is the value to be added to the calculated subscript to give the exact location of the element.

FA - First Address is the absolute address of the check word which stands just before the first element in the array.

D_n - are the "dope vector elements" which are only present if the array has more than one dimension.
Their use is explained by the following algorithm.

For an array with n dimensions the element with subscripts S_1, S_2, $S_3 \ldots S_n$ has the following address

$$<\text{absolute address of array element } (S_1,S_2,\ldots S_n)>=$$
$$(\ldots((S_n * D_n + S_{n-1}) * D_{n-1} + S_{n-2}) * D_{n-2} \ldots) * D_2 + S_1 + BA$$

For COMPLEX or REAL2 arrays the algorithm has the form

$$<\text{absolute address of double array element } (S_1,S_2,\ldots,S_n)>=$$
$$(2 * [(\ldots((S_n * D_n + S_{n-1}) * D_{n-1} + S_{n-2}) * D_{n-2} \ldots) * D_2 + S_1] + BA$$

Example:

The array element A(I,J,K) has the address
(K*D3+J)*D2+I+BA.

Checkword

The checkword at location FA has the following format.

```
          F3      FORM      18,18
                  F3        <length of array in machine words>,
                            <not used>
```

7.3.5.5 String Array Parameters.

The absolute data address (ADA)
is the start address of the string array descriptor.

The string array descriptor has the following format.

```
    Address

    ADA           <Relative string descriptor>
    ADA+1    )
    ADA+2    |
    ADA+3    |     Same as words ADA through ADA+5
    ADA+4    |     for ordinary arrays
    ADA+5    |
    ADA+6    )
```

The relative string descriptor has the following form

```
          F4      FORM      12,6,18
                  F4        <length>,<start>,<relative position>
```

where	<length>	is the number of characters in the string.
	<start>	is the start position of the string in the first word it occupies. S1=0 S2=1 S3=2 S4=3 S5=4 S6=5 (not 0 only for subarray elements)
	<relative position>	is the amount to be added to the address given in the string descriptor to get the address of the first word containing the string.

The address of an element is calculated in the same way as for ordinary arrays.

An element in a string array is a string descriptor

```
        F4      FORM      12,6,18
                F4        <length>,<start>,<address of string>
```

where <length> and <start> have the same meaning as above. In the case of a main string they will have the same values as well.

address of string   is the location of the first word used for the main string.

To find the address of the first word used for a substring, it is necessary to add the address of string to the relative position .

Example:

```
STRING ARRAY S1(7,S2(5,S3(4)),2:1:2,1:5)$
EXTERNAL SLEUTH PROCEDURE XYZ$
 .
 .
 .
XYZ(S1,S2,S3)$
```

Storage diagrams

ADA for S1

18	0	0
BA		FA
D2		

ADA for S2

9	1	1
BA		FA
D2		

ADA for S3

4	0	2
BA		FA
D2		

FA

SA = Start address

	10	
18	0	SA
18	0	SA+3
18	0	SA+6
18	0	SA+9
18	0	SA+12
18	0	SA+15
18	0	SA+18
18	0	SA+21
18	0	SA+24
18	0	SA+27

SA	S1(1,1:1,1)	S1(2,1:1,1)	S1(3,1:1,1)	S1(4,1:1,1)	S1(5,1:1,1)	S1(6,1:1,1)
	S1(7,1:1,1)	S1(8,1:1,1) S2(1,1:1,1)	S1(9,1:1,1) S2(2,1:1,1)	S1(10 S2(3	S1(11 S2(4	S1(12 S2(5
	S1(13 S2(6 S3(1	S1(14 S2(7 S3(2	S1(15 S2(8 S3(3	S1(16 S2(9 S3(4	S1(17	S1(18
SA+3	S1(1,1:2,1	S1(2				

7.4 Standard Procedures.

7.4.1 Available Procedures.

The following procedures are available for use without declaration. Also some identifiers with special meaning are listed.
These names are not reserved identifiers and may be redefined in any block.

X is used to mean the value of the first parameter,
Y the second.

Name	Number of Parameters	Types of Parameters	Result or Use	Type of Result
ABS	1	INTEGER REAL REAL2 COMPLEX	The absolute value of the parameter	INTEGER REAL REAL2 REAL
ALPHABETIC	1	STRING	TRUE if the string consists only of spaces or alphabetics (A-Z), FALSE otherwise.	BOOLEAN
ARCCOS	1	INTEGER REAL REAL2	arccos (X) arccos (X)	REAL REAL2
ARCSIN	1	INTEGER REAL REAL2	arcsin (X) arcsin (X)	REAL REAL2
ARCTAN	1	INTEGER REAL REAL2	arctan (X) arctan (X)	REAL REAL2
CARDS	0	-	To specify to the input routine that the device is the card reader or to the output routine that the device is the the card punch	

Name	Number of Parameters	Type of Parameters	Result of Use	Type of Result
CBROOT	1	INTEGER REAL	cube root of X	REAL
		REAL2	cube root of X	REAL2
		COMPLEX	cube root of X	COMPLEX
CHAIN	1	INTEGER	calls in link X in MAP	-
CLOCK	0	-	Present time of day in seconds since midnight. For example at 13:30 the result is 48600	INTEGER
COMPL	2	1. INTEGER REAL REAL2 2. INTEGER REAL REAL2	A complex number with the real part equal to X and the imaginary part equal to Y. <u>Example</u>: COMPL(1,2) gives the complex number <1.0,2.0>	COMPLEX
COS	1	INTEGER REAL	cos (X)	REAL
		REAL2	cos (X)	REAL2
		COMPLEX	cos (X)	COMPLEX
COSH	1	INTEGER REAL	cosh (X)	REAL
		REAL2	cosh (X)	REAL2
		COMPLEX	cosh (X)	COMPLEX
DISCRETE	2	1. REAL ARRAY 2. INTEGER	Drawing from a discrete (cumulative) distribution function (For full description see sec. 7.4.2)	
DRAW	2	1. REAL 2. INTEGER	TRUE with the probability X, FALSE with the probability 1-X (sec. 7.4.2)	BOOLEAN

Name	Number of Parameters	Type of Parameters	Result of Use	Type of Result
DRUM	0 or 1	INTEGER	Gives input/output routine access to relative address X of random drum. If X not specified then the next relative address available is used.	-
DRUMPOS	0	-	Gives next relative drum address	INTEGER
DOUBLE	1	INTEGER REAL	Value of type REAL2	REAL2
ENTIER	1	REAL REAL2	Largest integer I such that I $\leq$ X Example: ENTIER(-0.99) is -1	INTEGER
EOF	0 or 1	INTEGER REAL STRING	Used by WRITE and POSITION (See sec. 8.4.5) Only the first 6 characters of the string are used.	
EOI	0	-	Used by WRITE and POSITION (See sec. 8.4.6)	-
ERLANG	3	1. REAL 2. REAL 3. INTEGER	A drawing from the Erlang distribution with mean 1/X and standard devitation $1/X\sqrt{Y}$ (For full description see sec. 7.4.2)	REAL
EXP	1	INTEGER REAL	exp (X)	REAL
		REAL2	exp (X)	REAL2
		COMPLEX	exp (X)	COMPLEX

Name	Number of Parameters	Types of Parameters	Result or Use	Type of Result
HISTD	2	1. REAL ARRAY 2. INTEGER	A drawing from a histogram (For full description see sec. 7.4.2)	INTEGER
HISTO	4	1. REAL or INTEGER ARRAY 2. REAL or INTEGER ARRAY 3. REAL 4. REAL	To update a histogram according to observation (third parameter) with the weight the fourth parameter (For full description see sec. 7.4.2)	-
IM	1	COMPLEX	Imaginary part of the complex number X	REAL
INT	1	REAL REAL2 STRING	Value of type INTEGER	INTEGER
KEY	0 or 1	INTEGER	Used by WRITE and POSITION (See sec. 8.4.4) Only the first 6 characters of the string are used.	-
LENGTH	1	STRING	Number of characters in the string including blanks. <u>Example</u>: STRING S(42)$ LENGTH (S) has the value 42	INTEGER

Name	Number of Parameters	Types of Parameters	Result or Use	Type of Result
LINEAR	3	1. REAL ARRAY 2. REAL ARRAY 3. INTEGER	A drawing from a (cumulative) distribution using linear interpolation in a non-equidistant table, (for full description see sec. 7.4.2).	REAL
LN	1	INTEGER REAL	ln (X)	REAL
		REAL2	ln (X)	REAL2
		COMPLEX	ln (X)	COMPLEX
MARGIN	3 or 4	1. INTEGER 2. INTEGER 3. INTEGER 4. STRING	To change the vertical dimensions on a printer page (see sec. 8.8.5).	-
MAX	List of expressions (any number)	INTEGER REAL	Algebraic largest element of list Example: Value of MAX(FOR I=(1,1,99) DO I) is 99.0	REAL REAL
MIN	List of expressions (any number)	INTEGER REAL	Algebraic smallest element of list Example: Value of MIN (1.2,3.3,-8.6,-99.2,-4,0) is -99.2	REAL REAL
MOD	2	1. INTEGER REAL REAL2	If REAL or REAL2 then round x and y to nearest integer, then the expression X-ENTIER(X/Y)xY is computed.	INTEGER
		2. INTEGER REAL REAL2	Example: Value of MOD(-48,5) Is 2	

Name	Number of Parameters	Types of Parameters	Result or Use	Type of Result
NEGEXP	2	1. REAL 2. INTEGER	A drawing from the negative exponential distribution with mean 1/X (for full description see sec. 7.4.2).	REAL
NORMAL	3	1. REAL 2. REAL 3. INTEGER	A drawing from the normal distribution with mean X and standard deviation Y. (See sec. 7.4.2).	REAL
NUMERIC	1	STRING	TRUE if string has the form of an integer, FALSE otherwise.	BOOLEAN
POISSON	2	1. REAL 2. INTEGER	A drawing from the Poisson distribution (See sec. 7.4.2).	INTEGER
POSITION	special list	-	To position a tape (See section 8.8.3).	-
PRINTER	0	-	To assign the printer as device to the WRITE procedure	-
PSNORM	4	1. REAL 2. REAL 3. INTEGER 4. INTEGER	An approximate drawing from the normal distribution with mean X and standard deviation Y (See sec. 7.4.2)	REAL

Name	Number of Parameters	Types of Parameters	Result or Use	Type of Result
RANK	1	STRING	The field data equivalent of the first non-blank character of the string. Example: STRING S(12)$ S=' D'$ RANK(S) will have the value 9 ($D=11_8$).	INTEGER
RANDINT	3	1. INTEGER 2. INTEGER 3. INTEGER	A drawing of one of the integers between X and Y with equal probability (See description in sec. 7.4.2).	INTEGER
RE	1	COMPLEX	The real part of the complex number X.	REAL
READ	Special list	-	To bring input from a specified device	-
REWIND	1	TAPE	To rewind a tape	-
REWINT	1	TAPE	To rewind a tape and lock	-
SIGN	1	INTEGER REAL REAL2	Value of X / Value of SIGN (X): X>0 : 1 X=0 : 0 X<0 : -1 Example: Value of SIGN(128) is 1	INTEGER

Name	Number of Parameters	Types of Parameters	Result or Use	Type of Result
SIN	1	INTEGER REAL	sin (X)	REAL
		REAL2	sin (X)	REAL2
		COMPLEX	sin (X)	COMPLEX
SINH	1	INTEGER REAL	sinh (X)	REAL
		REAL2	sinh (X)	REAL2
		COMPLEX	sinh (X)	COMPLEX
SQRT	1	INTEGER REAL	$\sqrt{X}$	REAL
		REAL2	$\sqrt{X}$	REAL2
		COMPLEX	$\sqrt{X}$	COMPLEX
TAN	1	INTEGER REAL	tan (X)	REAL
		REAL2	tan (X)	REAL2
		COMPLEX	tan (X)	COMPLEX
TANH	1	INTEGER REAL	tanh (X)	REAL
		REAL2	tanh (X)	REAL2
		COMPLEX	tanh (X)	COMPLEX
TAPE	1	INTEGER STRING	To specify which tape or sequential drum file an input or output routine should use.	
UNIFORM	3	REAL REAL INTEGER	The value is uniformly distributed in the interval [X,Y>. (Sec.7.4.2).	REAL
WRITE	Special list	-	To send output to a specified device	-

7.4.2 Special Routine Descriptions.

Included in the run-time system of this ALGOL are many of the Random Drawing and some of the Data Analysis routines of SIMULA (O.J. Dahl, K. Nygaard: Simula. NCC. Sept.1967, ch. 7-8).

The following descriptions explain their uses and methods.

a) Pseudo-random Number Streams

All random drawing procedures of SIMULA use the same technique of obtaining basic drawings from the uniform distribution in the interval <0,1>.

A basic drawing will replace the value of a specified integer variable say, U, by a new value according to the following algorithm.

$$U_{i+1} = \text{remainder } ((U_i \times 5^{2p+1}) // 2^n),$$

where U_i is the i'th value of U.

It can be proved that, if U_0 is a positive odd integer, the same is true for all U_i, and the sequence U_0, U_1, U_2, --- is cyclic with the period 2^{n-2}. (The last two bits of U remain constant, while the other n-2 take on all possible combinations). In UNIVAC 1107/1108 we have n = 35. p is chosen equal to 6.

The real numbers $u_i = U_i \times 2^{-n}$ are fractions in the range <0,1>. The sequence u_1, u_2, --- is called a stream of pseudo-random numbers, and u_i (i = 1,2, ---) is the result of the i'th basic drawing in the stream U. A stream is completely determined by the initial value U_0 of the corresponding integer variable. Nevertheless it is a "good approximation" to a sequence of truly random drawings.

By reversing the sign of the initial value U_0 of a stream variable the antithetic drawings 1 - u_1, 1 - u_2, --- are obtained. In certain situations it can be proved that means obtained from samples based on antihetic drawings have a smaller variance than those obtained from uncorrelated streams. This can be used to reduce the sample size required to obtain reliable estimates.

b) Random Drawing Procedures

The following procedures all perform a random drawing of some kind. Unless otherwise is explicitly stated the drawing is effected by means of one single basic drawing, i.e. the procedure has the side effect of advancing the specified stream by one step. The necessary type conversions are effected for the actual parameters, with the exception of the last one. The latter must always be an integer variable specifying a pseudo-random number stream. All parameters except the last one and arrays are called by value.

1. Boolean procedure draw (a, U); real a; integer U;

   The value is true with the probability a, false with the probability 1 - a. It is always true if a $\geq$ 1, always false if a $\leq$ 0.

2. integer procedure randint (a, b, U); integer a, b, U;

   The value is one of the integers a, a + 1, ---, b - 1, b with equal probability. It is assumed that b $\geq$ a.

3. real procedure uniform (a, b, U); real a, b; integer U;

   The value is uniformly distributed in the interval $[a, b\rangle$. It is assumed that b > a.

4. real procedure normal (a, b, U); real a, b; integer U;

   The value is normally distributed with mean a and standard deviation b. An approximation formula is used for the normal distribution function:

   See M. Abramowitz & I.A. Stegun (ed): Handbook of Mathematical Functions, National Bureau of Standard Applied Mathematics Series no. 55, p. 952 and C. Hastings formula (26.2.23) on p. 933.

5. real procedure psnorm (a, b, c, U); real a, b; integer c, U;

   The value is formed as the sum of c basic drawings, suitably transformed so as to approximate a drawing from the normal distribution. The following formula is used:

$$a + b\left(\left(\sum_{i=1}^{c} u_i\right) - c/2\right)\sqrt{12/c}$$

This procedure is faster, but less accurate than the preceding one. c is assumed ≤ 12.

6. real procedure negexp (a, U); real a; integer U;

   The value is a drawing from the negative exponential distribution with mean 1/a, defined by -ln(u)/a, where u is a basic drawing. This is the same as a random "waiting time" in a Poisson distributed arrival pattern with expected number of arrivals per time unit equal to a.

7. integer procedure Poisson (a, U); real a; integer U;

   The value is a drawing from the Poisson distribution with parameter a. It is obtained by n+1 basic drawings, u_i, where n is the function value. n is defined as the smallest non-negative integer for which

$$\prod_{i=0}^{n} u_i < e^{-a}.$$

   The validity of the formula follows from the equivalent condition

$$\sum_{i=0}^{n} -\ln(u_i)/a > 1,$$

   where the left hand side is seen to be a sum of "waiting times" drawn from the corresponding negative exponential distribution.

   When the parameter a is greater than 20.0, the value is approximated by integer (normal (a,sqrt(a),u)) or, when this is negative, by zero.

8. real procedure Erlang (a, b, U); value a, b; real a, b; integer U;

   The value is a drawing from the Erlang distribution with mean 1/a and standard deviation $1/(a\sqrt{b})$. It is defined by b basic drawings u_i, if b is an integer value,

$$-\sum_{i=1}^{b} \frac{\ln(u_i)}{a \cdot b},$$

and by c+1 basic drawings u_i otherwise, where c is equal to entier (b),

$$-\sum_{i=1}^{c} \frac{\ln(u_i)}{a \cdot b} - \frac{(b-c) \quad \ln(u_{c+1})}{a \cdot b}$$

Both a and b must be greater than zero.

9. <u>integer</u> <u>procedure</u> discrete (A, U); <u>array</u> A; <u>integer</u> U;

   The one-dimensional array A, augmented by the element 1 to the right, is interpreted as a step function of the subscript, defining a discrete (cumulative) distribution function. The array is assumed to be of type <u>real</u>.

   The function value is an integer in the range [lsb, usb+1], where lsb and usb are the lower and upper subscript bounds of the array. It is defined as the smallest i such that A(i)> u, where u is a basic drawing and A (usb+1) = 1.

10. <u>real</u> <u>procedure</u> linear (A, B, U); <u>array</u> A, B; <u>integer</u> U;

   The value is a drawing from a (cumulative) distribution function F, which is obtained by linear interpolation in a non-equidistant table defined by A and B, such that A (i) = F(B(i)).

   It is assumed that A and B are one-dimensional <u>real</u> arrays of the same length, that the first and last elements of A are equal to 0 and 1 respectively and that A (i) $\geq$ A (j) and B (i) > B (j) for i > j.

11. <u>integer</u> <u>procedure</u> histd (A, U); <u>array</u> A; <u>integer</u> U;

   The value is an integer in the range [lsb, usb], where lsb and usb are the lower and upper subscript bound of the one-dimensional array A. The latter is interpreted as a histogram defining the relative frequencies of the values.

This procedure is more time-consuming than the procedure discrete, where the cumulative distribution function is given, but it is more useful if the frequency histogram is updated at run-time.

12. procedure histo (A, B, c, d); array A, B; real c, d; will update a histogram defined by the one-dimensional arrays A and B according to the observation c with the weight d. A (i) is increased by d, where i is the smallest integer such that c ≤ B (i). It is assumed that the length of A is one greater than that of B. The last element of A corresponds to those observations which are greater than all elements of B. The procedure will accept parameters of any combination of real and integer types.

7.4.3 Tranfer Functions.

Transfer functions are those functions used to "transfer" a value of one type to another type. These functions are evoked automatically by the compiler whenever necessary. In some cases, they may be called explicitly. Note that transfer functions are not evoked automatically when the formal and actual types for array identifiers are not the same.

Type of variable	Transferred to type	Function used
INTEGER	REAL	Implicit
	REAL2	Implicit
	STRING	Implicit
	COMPLEX	COMPL(X,0) or Implicit
REAL	INTEGER	INT(X) or implicit
	REAL2	Implicit
	COMPLEX	COMPL(X,0) or Implicit
REAL2	INTEGER	INT(X) or implicit
	REAL	Implicit
	COMPLEX	COMPL(X,0) or Implicit
COMPLEX	REAL	RE(X)
		IM(X)
STRING	INTEGER	INT(X) or implicit

8 INPUT/OUTPUT

8.1 Introduction

Form

All input/output statements are of the form

```
<I/0 procedure>(<device>,<format>,<modifierlist>,
                 <input/output list>,<label list>)$
```

This chapter is organized in such a way that the parameters <device>,<modifier list>,<label list>,<format> and <input/output list> are described in separate sections.

Each of the procedures is then described in terms of the parameters it requires.

Example:

```
BEGIN FORMAT FORM1 (A,3R10.2)$
REAL X,Y,Z$
ARRAY ARRY (1:200)$
WRITE (TAPE('A'),EOF('ABC'),LABL1,ARRY)$
READ  (CARDS,FORM1,LABL2,LABL2,X,Y,Z)$
READ  (CARDS,X,Y,TAPE(12),ARRY)$
COMMENT MORE THAN ONE DEVICE ALLOWED$
```

Method

The available input/output procedures are:

Procedure	Section
READ	8.9
WRITE	8.8
POSITION	8.10
REWIND REWINT	8.11

Classed as tape operations

8.2 Parameters to Input/Output Statements

Number of parameters

The procedures allow a variable number of parameters. In the simplest case only the input/output list needs to appear. The other parameters are then automatically supplied by the compiler. See sec. 8.8.

Example:

```
FORMAT F(10I12,A1)$
INTEGER ARRAY A(-6:3)$
WRITE (A)$
WRITE (PRINTER,F,A)$  COMMENT THESE TWO ARE THE SAME$
WRITE (CARDS,A)$
WRITE (CARDS,F,A)$  COMMENT THESE TWO ARE THE SAME$
```

Order of parameters

The order of parameters is very important. In general, all statements should have their parameters in the order given by the form of sec. 8.1.

If this order is not observed, the following rules hold.

a) Labels may come anywhere and need not to be together. However, their order is important. (See section 8.5, label list).

b) If device is not before the input/output list, then the device is assumed to be implied device. (See section 8.3.3, implied device).

c) The insertion of more device calls in an I/O statement changes the device.
   Example:

```
ARRAY A(0:500)$
WRITE (A,TAPE('B'),A)$
COMMENT WILL WRITE ARRAY A ON THE PRINTER AND ON
THE MAGNETIC TAPE ASSIGNED AS B $
```

d) Modifiers may be placed where desired. That is, KEY will usually come before the output list and EOF after it, but notice the following example.

Example:

```
ARRAY A(0:500),B(0:300)$
WRITE(TAPE('B'),KEY('A'),A)$
WRITE(TAPE('B'),EOF('A'),KEY('B'),B,EOI)$

COMMENT THE TAPE WILL HAVE
1. KEY RECORD WITH IDENTIFICATION 'A'
2. THE VALUES OF THE ARRAY A
3. EOF RECORD WITH IDENTIFICATION 'A'
4. KEY RECORD WITH IDENTIFICATION 'B'
5. THE VALUES OF THE ARRAY B
6. AN EOI MARKER$
```

e) Formats must come before the input/output list to which they apply. If a list comes before a format parameter has been specified, then the format is taken to be implied or free format.

Example:

```
INTEGER I,J,K$
REAL    X,Y,Z$
FORMAT  F(3D10.6,A1)$
I=123$  J=456$  K=789$
WRITE (I,J,K,F,I,J,K)$
COMMENT WILL PRODUCE THE FOLLOWING PRINT LINES$
         123          456          789
123.00000   456.00000   789.00000
```

f) Formats must come after the device to which they apply.

g) Input/output lists have their position determined by the fact that they must conform to the above rules.

8.3 Devices

8.3.1 Possible Devices

The possible devices are

Device	Section
(implied)	8.3.3
CARDS	8.3.4
PRINTER	8.3.5
TAPE	8.3.6
DRUM	8.3.7
CORE	8.3.8

8.3.2 Actual Devices

Device	Actual device with READ	Actual device with WRITE	Actual device with POSITION, REWIND, REWINT
(implied)	Card reader	Line printer	Not allowed
CARDS	Card reader	Card punch	Not allowed
PRINTER	Not allowed	Line printer	Not allowed
TAPE	Tape unit or drum file specified	Tape unit or drum file specified	Tape unit or drum file specified
DRUM	Random access drum file	Random access drum file	Not allowed
CORE	The string which is parameter	The string which is parameter	Not allowed

Examples:

```
INTEGER I$
READ (CARDS,I)$
READ(I)$  COMMENT ARE THE SAME$
```

8.3.3 Implied Devices

Use

For reading cards or printing.

Form

The device parameter is left out.

Action with READ

Same as for device CARDS.

Action with WRITE

Same as for device PRINTER.

Restrictions

i) Cannot be used with TAPE operations.
ii) On input only 80 columns may be read from a card.
iii) On output only 132 columns may be printed.

Example:

```
INTEGER A,B,C,D$
FORMAT F1(A,3(I12,X10))$
READ (F1,A,B,C)$
COMMENT WILL READ CARDS$
```

8.3.4 Device CARDS

Use

For reading or punching cards.

Form

CARDS

Action with READ

The card reader is assigned as the device for the procedure READ to use for input.

Note: If a format is specified, no new card is read until an A phrase (activate) is met in a format or a format extends beyond column 80 of the current card. The very first data card, however, will be read automatically in the absence of an A-phrase.

Re-reading

Reading card images over again is possible by using a format without an activate phrase.

Example:

```
BEGIN
  COMMENT READ THE SAME CARD IN THREE DIFFERENT WAYS$
  ARRAY A,B,C(1:5)$
  FORMAT F1(A,5I5),
         F2(J1,5I1),
         F3(J1,5I2)$
  COMMENT NOTE THAT J-PHRASE MUST BE USED TO START
  AT COLUMN ONE$
  READ (F1,A,F2,B,F3,C)$
END$
```

Data Card

```
1234567891011121314151617
```

Action:

At the end the arrays will have the following values:

A(1)	12345.0	B(1)	1.0	C(1)	12.0
A(2)	67891.0	B(2)	2.0	C(2)	34.0
A(3)	1112.0	B(3)	3.0	C(3)	56.0
A(4)	13141.0	B(4)	4.0	C(4)	78.0
A(5)	51617.0	B(5)	5.0	C(5)	91.0

Action with WRITE

The card punch is assigned as the device for the procedure WRITE to use for output.

Example:

```
FORMAT F(I12,A1)$
INTEGER I$
I=-8523$
WRITE (CARDS,F,I)$
COMMENT WILL PUNCH ONE CARD WITH -8523 IN COLUMNS 8
THROUGH 12$
```

Restrictions

i) Cannot be used with the tape operations.
ii) On both input and output there is a maximum length of 80 columns.

8.3.5 Device PRINTER

Use

For printing on printer.

Form

PRINTER

Action with WRITE

The line printer is assigned as the device for the procedure WRITE to use for output.

Note: If a format is specified, no line is printed until an activate (A) phrase is processed. The A-phrase may be delayed until a later WRITE-statement.

Example:

```
INTEGER I,J$
WRITE (PRINTER, <<I15,A1,I6>>,I,J)$
COMMENT J IS NOT PRINTED$
WRITE (PRINTER,<<I10,A1>>,I)$
COMMENT PRINTS J AND I ON THE SAME LINE$
```

Restrictions

i) A run-time error is caused if PRINTER is used with READ or the tape operations.
ii) One line has 132 columns.

Example:

```
ARRAY A(-5:6)$
INTEGER X,Y$
FORMAT F1(12(I11,X1),A1)$
WRITE (PRINTER,F1,FOR I=(-5,1,5) DO A(I))$
```

8.3.6 Device TAPE

Use

For doing operations with magnetic tapes or sequential drum files.

Form

TAPE (<parameter>) where <parameter> can be

i) non-negative integer constant or expression which is the index in the range 0 to 20 to the Y$TTAB table given below.

ii) string in which the first character is the logical unit designation for an assigned magnetic tape.

Examples:

```
ARRAY A(0:500)$  INTEGER I$
I=0$
WRITE (TAPE('A'),A)$
WRITE (TAPE (0),A)$
WRITE (TAPE (I),A)$
COMMENT PROVIDE ALL THE SAME ACTIONS$
```

Meaning of parameters

The parameter is an index to an installation defined Y$TTAB table.

Note: It is possible for the user to supply his own Y$TTAB table - perhaps redefining some of the drum areas. However this should only be done with the help of the systems programmer for his installation.

The following is the implemented Y$TTAB table.
Note that the drumfiles occupy the same area as the PCF, and processor scratch.

Y$TTAB

Parameter		Meaning	
Integer	String		
0	'A'	Use magnetic tape assigned as A	
1	'B'	assigned as B	
2	'C'	assigned as C	
3	'D'	assigned as D	
4	'E'	assigned as E	
5	'F'	assigned as F	
	Tape simulating files	Drum layout	
6	↑	Whole	
7	Not	1st half	
8		2nd half	
9	Allowed	1st quarter	
10		2nd quarter	
11		3rd quarter	
12		4th quarter	
13		1st eighth	
14		2nd eighth	
15		3rd eighth	
16		4th eighth	
17		5th eighth	
18		6th eighth	
19		7th eighth	
20	↓	8th eighth	

Action with READ and WRITE

Assign the specified magnetic tape unit or sequential drum file to be used by READ or WRITE for input or output.

Example:

```
REAL2 ARRAY D(0:400)$ INTEGER I$
READ (TAPE(20),FOR I=(1,1,320) DO D(I))$
WRITE(TAPE('A'),FOR I=(1,1,300) DO D(I))$
```

Action with REWINT

If the parameter refers to a magnetic tape then this tape is rewound and released so that it can no longer be used.

If the parameter refers to a sequential drum file, then the current position of this file is reset to the starting position.

Example:

```
INTEGER I$
FOR I=(0,1,20) DO REWINT (TAPE(I))$
COMMENT WILL REWIND AND RELEASE MAGNETIC TAPES 'A'
THROUGH F AND RESET TO THE START DRUM FILES 6
THROUGH 20$
```

Action with REWIND

For magnetic tapes, the tape is rewound but not released so that it may be used again.

The action for sequential drum files is the same as for REWINT.

Example:

```
BOOLEAN DRUMORTAPE$
DRUMORTAPE=TRUE$
REWIND (TAPE(IF DRUMORTAPE THEN 0 ELSE 6))$
COMMENT WILL REWIND TAPE ASSIGNED AS A$
```

Action with POSITION

The specified magnetic tape or sequential drum file is assigned to the procedure POSITION. It will then be searched according to certain parameters. This operation is covered in section 8.10.

Example:

```
POSITION (TAPE('D'),EOF)$
```

Restrictions

i) The sequential drum files can only be accessed in a serial manner. If random access is required, device DRUM must be used.

ii) Device TAPE does not allow READ or WRITE to use a format. To write formatted output one can use WRITE (CORE(S),...) and then output the resulting string.

iii) The input list (see section 8.7) used with device TAPE must have its number of elements less than or equal to the number of elements in the output list which produced the record being read.

If the number is greater a run-time error occurs.

If the input list is smaller than the output list then the remainder of the record is lost.

iv) If the integer expression used as parameter to TAPE has a value greater than 20 or less than 0, a runtime error occurs.

v) The expression used as parameter to TAPE must not be a type procedure.

vi) The format of records for device TAPE are compatible with both UNIVAC ALGOL and FORTRAN.

Examples:

```
ARRAY A,B (1:500)$
INTEGER I$
FORMAT F(10R12.4,A1)$
READ (TAPE(6),A)$   COMMENT TRANSFERS 500 WORDS FROM
THE DRUM FILE KNOWN AS TAPE(6) TO THE ARRAY A$
WRITE (TAPE('E'),FOR I=(1,1,250) DO B(I))$
WRITE (TAPE('E'),FOR I=(251,1,500) DO B(I))$
REWIND (TAPE('E'))$
READ (TAPE('E'),FOR I=(1,1,200) DO A(I))$
READ (TAPE('E'),FOR I=(251,1,500) DO A(I))$
COMMENT A(201) TO A(250) WILL NOT BE CHANGED WHILE THE
VALUES B(201) TO B(250) TO B(250) ON TAPE WILL BE LOST$
```

8.3.7 Device DRUM

Use

To use the random access drum file.

Form

DRUM (<aritmetic expression>) or DRUM

i) The arithmetic expression indicates the relative address of that part of the drum which has been set aside for random access.

Example:

```
REAL X,Y,Z$
INTEGER I$
I=50$
WRITE (DRUM(I),X,Y,Z)$
COMMENT WILL WRITE THE VALUES OF THE VARIABLES X,
Y,Z IN RELATIVE ADDRESSES
50,51 AND 52 OF THE DRUM$
```

ii) If no parameter is given then the parameter refers to next relative address of the random drum file.

Example:

```
COMMENT THIS STATEMENT COMES IMMEDIATELY AFTER THE
ONES ABOVE$
READ (DRUM,X,Y,Z)$
COMMENT VALUES ARE TRANSFERRED TO X,Y,Z FROM
RELATIVE ADDRESSES 53, 54 AND 55$
```

iii) The drum address may be set to a specified position prior to a READ/WRITE-statement by the statement:

```
DRUM(<arithmetic expression>)$
```

The DRUMPOS procedure

This procedure obtains the next relative drum address.

Example:

```
WRITE (DRUM(100),X,Y,Z);
I=DRUMPOS;
COMMENT I NOW HAS THE VALUE 103;
```

Action with WRITE

The values of the variables of the output list are transferred to consecutive positions in the random drum file area starting at the relative address specified by the parameter given to the procedure DRUM. If no parameter is given then the start is the next relative address.

Action with READ

The values of the consecutive positions in the random drum file starting with the relative address specified by the parameter to DRUM are transferred to the input list variables. If no parameter is given then the start is the next relative address.

Restrictions

i) DRUM may not be used with the tape operations.

ii) To determine the relative address after a WRITE using DRUM it is necessary to know the following lengths.

Variable Type	Length in words
INTEGER	1
REAL	1
BOOLEAN	1
REAL2	2
COMPLEX	2
STRING of k characters	ENTIER((k+5)/6)+1
SUBSTRINGS of length k which start at character p in a word ($0 \leq p \leq 5$)	ENTIER ((p+k+5)/6)+1

iii) DRUM and TAPE (6 through 20) share an area on drum. The user should ensure that they do not overwrite each other. They both overwrite the PCF area.

Examples:

```
BEGIN
  INTEGER I$
  REAL R$
  BOOLEAN B$
  REAL2 D$
  COMPLEX C$
  STRING S(15)$
  WRITE (DRUM(1),I,R,B,D,C,S)$
  COMMENT THE NEXT RELATIVE DRUM ADDRESS IS 12$
END$
```

Drum Notes

i) Parameters in a list are automatically placed in consecutive locations on the drum.

Example:

```
    WRITE (DRUM(0),A,B,C,-----)
```

and

```
    WRITE (DRUM(0),A,DRUM(1),B,DRUM(2),C,-----)
```

do exactly the same operation - BUT the first case is much faster.

ii) Because of the mechanism used for writing drum - writing backwards on drum is extremely inefficient.

Example:

```
    WRITE (DRUM(25),Z,DRUM(24),Y,DRUM(23),X-----)$
    COMMENT - IS VERY SLOW$
```

iii) Arrays are normally transferred without being decomposed into their elements. For this reason, statements which decompose an array are very inefficient in comparison.

Example:

```
ARRAY A(1:500)$  INTEGER I$
WRITE (DRUM,A)$  COMMENT IS VERY FAST$
WRITE (DRUM,FOR I=1,1,500 DO A(I))$
FOR I=1,1,500 DO WRITE(DRUM,A(I))$
COMMENT THE LAST TWO STATEMENTS ARE VERY SLOW$
```

8.3.8 Device CORE

Use

To allow editing to and from a string without using an external device.

Form

CORE (<string expression>)

Action with WRITE

The output list is edited according to the given or implied format into the string supplied as the parameter to CORE.

Example:

```
BEGIN
  STRING S(24)$
  FORMAT F(6I4,A)$
  INTEGER ARRAY A(1:6)$
  INTEGER I$
  FOR I=(1,1,6) DO A(I)=I$
  WRITE(CORE(S),F,A)$
  COMMENT WILL CAUSE S TO BE FILLED AS IF THE
  FOLLOWING ASSIGNMENT HAD TAKEN PLACE
  S='   1   2   3   4   5   6'$
END$
```

Action with READ

The string is edited according to the given or implied format and the values assigned to the input list.

Example:

```
BEGIN
  STRING S(14)$  INTEGER I$  REAL R$
  FORMAT F(A,D12.2,I2)$
  S='  1234.5678421'$
  READ (F,CORE(S),R,I)$
  COMMENT R NOW HAS THE VALUE 1234.56784 AND I
  HAS THE VALUE 21$
END$
```

Restrictions

i) CORE cannot be used with the tape operations.

ii) On input (READ) only 80 characters may be edited.

iii) On output (WRITE) only 132 characters may be edited.

iv) The entire string is used by CORE.

Example:

```
STRING S(30)$
S(27,3)='ABC'
WRITE (CORE(S),1,2)$
COMMENT THE 'ABC' HAS BEEN CLEARED TO BLANKS$
```

v) Note that nothing is transferred to or from the string until the activate (A) phrase is reached in the format specified.

vi) If no format is specified the rules for free format (See Section 5.3) are applied.

8.4 Modifier List

Use

The modifier list contains directions as to the type of markers to be used with device TAPE.

8.4.1 Possible Modifiers

Modifier	Section
EOF	
EOF (<parameter>)	
-EOF	8.4.5
-EOF (<parameter>)	
KEY	
KEY (<parameter>)	
-KEY	8.4.4
-KEY (<parameter>)	
EOI	8.4.6
-EOI	
<integer expression>	8.10

8.4.2 General description

Action with WRITE

The modifier list contains a directive to output a certain marker which later can be searched for using action POSITION.

Action with POSITION

The modifier list contains the marker to be searched for.

8.4.3 Restrictions

The modifier list cannot be used with the operations READ, REWIND or REWINT.

Modifiers can only be used with device TAPE.

Certain tape units cannot be positioned backward.

TYPE OF TAPE UNIT	CAN BE POSITIONED BACKWARDS
II A	YES
III A	YES
III C	NO
IV C	NO
VI C	NO
VIII C	YES

Violating this rule causes a run-time error.

8.4.4 Modifier KEY

Use

To specify that a KEY record with a certain identification is to be output or searched for.

Form

KEY (<parameter>) or KEY
-KEY (<parameter>) or -KEY

The parameter can either be an arithmetic expression or a string expression. When the parameter is a string, only the first six characters are used. If the string is shorter, it is space filled up to six characters.

The minus (-) sign specifies the backwards direction when used with POSITION. It has no meaning for WRITE.

Note that KEY means the same as KEY (0)
-KEY means the same as -KEY (0)

Example:

```
WRITE (TAPE(0),KEY('ABCDEF'))$
WRITE (TAPE(0),KEY('ABCDEFGHK'))$
COMMENT WILL PROCEDURE TWO IDENTICAL KEY RECORDS$
```

Example:

```
POSITION (TAPE('A'),KEY)$
POSITION (TAPE('A'),KEY(0))$
COMMENT HAVE THE SAME MEANING$
```

Action with WRITE

A KEY record with its identification given by the parameter is output on the tape or sequential drum file.

Example:

```
INTEGER I,J,K,L,M$
WRITE(TAPE('F'),I,J,K,L,M,KEY(I))$
COMMENT THE KEY RECORD COMES AFTER THE RECORD$
REWIND (TAPE('F'))$
READ (TAPE('F'),I,J,K,L,M)$
COMMENT WILL READ THE VALUES INTO I,J,K,L,M IGNORING
THE KEY RECORD$
```

Action with READ

Key records are ignored.

Action with POSITION

For more information see section 8.10.

If no minus sign (-) then the action is to search forward until a KEY record with the given identification is found.

If there is a minus sign (-) then the action is to search backward (only on certain tape units and not on sequential drum files) until the KEY with the specified identification is found.

KEY records are ignored when positioning to EOF or EOI.

Example:

```
BOOLEAN B$
B = TRUE$
POSITION (TAPE(15),KEY (IF B THEN 10 ELSE 15),
                                         KEYNOTFOUND)$
COMMENT WILL SEARCH FORWARD FOR THE KEY RECORD WITH
IDENTIFICATION 10. IF THIS RECORD IS NOT FOUND, THEN
THE PROGRAM WILL JUMP TO THE STATEMENT WITH THE LABEL
KEYNOTFOUND$
```

For more information on labels in a POSITION see section 8.5.

Example:

```
ARRAY A(0:500)$
WRITE (TAPE('E'),EOF('END'),A)$
COMMENT WILL WRITE THE EOF RECORD WITH IDENTIFICATION
'END', AND THEN THE ARRAY A$
```

8.4.5 Modifier EOF

Use

To specify that an EOF (end of file) record with a certain identification is to be output or searched for.

Form

EOF (<parameter>) or EOF
-EOF (<parameter>) or -EOF

The parameter can either be an arithmetic expression or a string. When the parameter is a string, only the first six characters are used. If the string is shorter, it is space filled up to six characters.

The minus sign (-) specifies that the search is to be performed in a backwards direction when use with POSITION. It has no meaning for WRITE.

Note that EOF means the same as EOF (0)
        -EOF means the same as -EOF (0).

Action with WRITE

An EOF record with its identification given by the parameter is output on the tape or sequential drum file. A minus sign has no meaning.

Example:

```
ARRAY A(0:500)$
WRITE (TAPE('E'),A,EOF('END'))$
COMMENT WILL WRITE OUT THE RECORD CONTAINING THE
VALUES OF A AND THEN THE EOF RECORD WITH
IDENTIFICATION WORD 'END'$
```

Action with READ

If the READ operation encounters an EOF record, it will exit via a label in its label list if such a list exists. See section 8.5.

The modifier EOF must not be placed in a READ list.

Action with POSITION

If there is no minus sign (-), then the action is to search forward until an EOF record with the given identification is found.

If there is a minus sign (-), then the action is to search backward (only on certain units) until the EOF record with the specified identification is found.

Note: When positioning backwards, the positioning goes to the front of the EOF record so that the next READ action will encounter the EOF record.

Example:

```
ARRAY A(0:12)$
POSITION (TAPE(4),-EOF)$
READ (TAPE(4),EOFLB,A)$
COMMENT WILL JUMP TO THE STATEMENT WITH THE LABEL
EOFBL SINCE AN EOF RECORD WAS READ INSTEAD OF A
RECORD WITH THE VALUES FOR A$
```

EOF records are ignored when positioning to EOI.

8.4.6 Modifier EOI

Use

To specify that an EOI (end of information) record is to be output or searched for.

Form

EOI or -EOI
where the minus sign (-) indicates that search is to be performed in a backwards direction, when used with POSITION. It has no meaning for WRITE.

Action with WRITE

An EOI record is output.

Example:

```
COMPLEX ARRAY C(-4:200)$
WRITE (TAPE(5),C,EOI)$
COMMENT WILL WRITE ARRAY C TO TAPE AND THEN PLACE
AN EOI MARKER$
```

Action with READ

If the READ operation encounters an EOI marker, it will exit via a specific label in its label list, if such a list exists. See section 8.5.

Action with POSITION

The file is positioned in the indicated direction, past the first EOI record found.

8.5 Label List

Use

The label list allows the user to specify where he would like his program to go to if certain conditions occur during the input or output operation. If the operation ends normally, exit is made to the next statement, otherwise it is a run-time error.

Form

A label list consists of from zero to three labels together or scattered througout the parameter list to the input/output procedure. Their order is important. An input list may have three labels, an output list only one.

8.5.1 Action with READ when Device is Implied or CARDS

Number of labels	Action when EOF card read	Action when another control card read	Action when an error occurs including input or format errors
0	Terminate program	Terminate program	Terminate program
1	Jump to this label	Jump to this label	Terminate program
2	Jump to first label	Jump to second label	Terminate program
3	Jump to first label	Jump to second label	Jump to third label

8.5.2 Action with READ when Device is TAPE

Number of labels	Action when EOF record read	Action when EOI record read	Action when an error occurs
0	Terminate program	Terminate program	Terminate program
1	Jump to this label	Jump to this label	Terminate program
2	Jump to first label	Jump to second label	Terminate program
3	Jump to first label	Jump to second label	Jump to third label

8.5.3 Action with READ or WRITE when Device is DRUM

Number of labels	READ		WRITE	
	When address beyond random drum limits	When a drum read error occurs	When address beyond random drum limits	When a drum write error occurs
0	Terminate program	Terminate program	Terminate program	Terminate program
1	Jump to this label	Terminate program	Jump to this label	Jump to this label
2	Jump to second label first label ignored	Terminate program	Only one label	
3	Jump to second label first label ignored	Jump to third label	allowed with WRITE	

8.5.4 Action with READ or WRITE when Device is CORE

The only errors that can occur when using CORE, are format errors in reading. If no third label is given, the program is terminated. Otherwise exit is made to the third label ignoring other labels.

8.5.5 Action with WRITE when Device is implied, PRINTER or CARDS

All errors other than editing errors terminate the program. Editing errors cause a warning message, but the program continues.

8.5.6 Action with WRITE when Device is TAPE

Number of labels	Action on end of tape or end of sequential drum file	Action on tape error
0	Terminate program	Terminate program
1	Jump to this label	Jump to this label

8.5.7 Action with POSITION - only allowed Device is TAPE

See table on next page.

Example:

```
BEGIN
  COMMENT STOP READING DATA CARDS WHEN EOF CARD READ$
  INTEGER ARRAY A(0:1000)$ INTEGER I$
L0: READ (CARDS,A(I),L1,L2,L3)$
    I=I+1$ GO TO L0$
L3: WRITE ('ERROR IN CARD',I)$  GO TO L0$
L2: WRITE ('EOF CARD MISSING')$ GO TO STOP$
L1: WRITE ('ALL CARDS READ')$

STOP: END$
```

Action with POSITION

POSITION parameter	KEY or arithmetic expression			EOF		EOI
Tape contents	EOF	EOI	End of type type error	EOI	End of type, type error	End of tape, tape error
Number of labels						
0	Terminate program	Terminate program	Terminate program	Terminate program	Terminate program	Terminate program
1	Jump to label	Jump to label	Terminate program	Jump to label	Terminate program	Terminate program
2	Jump to first label	Jump to second label	Terminate program	Jump to second label, ignore first label	Terminate program	Terminate program
3	Jump to first label	Jump to second label	Jump to third label	Jump to second label, ignore first label	Jump to third label	Jump to third label ignore first and second

8.6 Format List

Use

The format list is a means of specifying how values should be edited.

Form

The format list may have any number of formats. Each format should come before the input or output list to which it applies.

Each format may have one of the three following forms.

Name	Section
Implied or free format	8.6.1
Declared format	8.6.2
Inline format	8.6.3

Restrictions

The devices TAPE and DRUM do not allow format lists. A run-time error is caused if an attempt is made to use a format with these devices.

8.6.1 Implied or Free Format

a) Form

No format is specified before an input/output list.

b) Action with READ

80 character images are input at a time, usually from punched cards, and for all devices, which allow formatted input, 80 characters are brought into a "read buffer" - which is an area in core from which editing can be done.

Values are separated by one or more blanks or end of card. Within a string end of card is ignored.

The characters encountered are scanned and converted into a value according to their form. The type of value is determined by the rules for constants as described in section 4.1.

Exceptions:

In real constants comma (,) or the letter E may be substituted for & as the power of ten symbol.

Complex constants should appear as two reals. (<,> must not be used).

Example:

Constant	Would be edited as type
123	INTEGER
TRUE	BOOLEAN
1.24,-3	REAL
1.2483212145	REAL2
'THIS IS A STRING'	STRING
1.245 3.217	COMPLEX

If the type of the value thus edited does not match the type of the list element to which it is to be assigned, a transfer function (if available) is invoked. If the types match, the values is assigned directly to the list element.

At the end of the image or when an asterisk (*) outside of string quotes is met, the next image is input.

The action ends when all elements in the input list have had values assigned to them. Any further information in the read buffer is lost since the next READ starts with a new image.

Examples:

```
BEGIN
  ARRAY X,Y(1:5,1:2)$
  REAL A,B$
  COMPLEX C$
  INTEGER W$
  READ(A,B,C,W,X,Y)$
END$
```

Data cards:

```
-7.2    .099   1.0 3.5    362236
1  2  3  4  5  6  * NOTE THAT ARRAYS ARE READ BY COLUMN

2.4   3.5   8.6   9.2   5.562,-4   4.398,-3
1.862,-1   12.842   18.623   1.5   1.6   1.7   1.8   1.9   2.0
```

Values after read is performed		
Variable	Has the value	Explanation
A	-7.2	
B	.099	
C	1.0+i*3.5	
W	362236	
X(1,1)	1.0	Shift to next card since
X(2,1)	2.0	not all list elements filled
X(3,1)	3.0	A transfer function is
X(4,1)	4.0	used here
X(5,1)	5.0	All characters after an
X(1,2)	6.0	* are ignored
X(2,2)	2.4	
X(3,2)	3.5	Arrays are decomposed by column
X(4,2)	8.6	
X(5,2)	9.2	
Y(1,1)	.0005562	
Y(2,1)	.004398	
Y(3,1)	.1862	
Y(4,1)	12.842	
Y(5,1)	18.623	

continued next page

	Values after read is performed	
Variable	Has the value	Explanation
Y(1,2)	1.5	
Y(2,2)	1.6	
Y(3,2)	1.7	
Y(4,2)	1.8	
Y(5,2)	1.9	
		The value 2.0 is not assigned to any variable but is lost

Example:

```
BEGIN
  STRING S(24)$
  INTEGER I,J,K,L,M,N$
  S='1  -2.1  3.5  8  4  6 '$
  READ (CORE(S),I,J,K,L,M,N)$
END$
```

Values after read is performed	
Variable	Value
I	1
J	-2
K	4
L	8
M	4
N	6

c) Action with WRITE

The action of WRITE is to evaluate the expressions in the order they appear in the output list and then edit the values according to the following rules.
(The format phrases used are described in section 8.6.3).

Type	Format phrase used
INTEGER	I12
BOOLEAN	X1,B11
REAL	R12.5
REAL2	R12.5
COMPLEX	2R12.5
STRING of length w	Sw,Xm - where m is the number of blanks required to fill out a multiple of 12 columns.

Example:

```
INTEGER I$    BOOLEAN N$    REAL R$
REAL2 D$      COMPLEX C$    STRING S(26)$
FORMAT F(S6,X6,I12,X1,B11,R12.5,R12.5,2R12.5,S26,
        X10,A1)$
STRING CONSTANT(6)$
I = 123$    B = TRUE$    R = 1.321&-2$
D = 1234.6789012$
C = <11.2,-12.4>$
S = 'IS THE WAY THE RESULTS ARE'$
CONSTANT =  'START'$
WRITE ('START',I,B,R,D,C,S)$
WRITE (F,CONSTANT,I,B,R,D,C,S)$
COMMENT WILL PRODUCE SIMILAR PRINTOUTS$
END$
```

8.6.2 Inline Format

Form

A list of format phrases enclosed between the delimiters << >> may be a parameter in the format list.

Example:

```
WRITE (<<3I3,A1>>,I,J,K)$
```

8.6.3 Declared Format

a) Purpose

A specific sequence of phrases is declared and an identifier attached which can be used in the format list.

b) Form

Format <identifier>(<list of format phrases>),
<identifier>(   ),.....$

Example:

```
FORMAT F1(X10,D7.2,X5,R17.8,A1.1),
       F2(A,B6,S10,I5,X2,N4)$
```

8.6.4 Format Phrases with WRITE

a) Use

Format phrases are used with WRITE to specify the output form of each parameter as well as the exact position for the placement of the value of the parameter.

b) Form of a format phrase

Qw.d
or $Q(E_1,E_2)$

where Q represents one of the letters given below.

E_1 must be a positiv arithmetic expression with the same meaning as the positiv integer constant w. The meaning and restrictions are given in the table below.

E_2 must be a positiv arithmetic expression with the same meaning as the positiv integer constant d. The meaning is given i the table below.

c) Available format phrases, meaning and restrictions with WRITE

The print buffer is a string of 132 characters for devices implied, PRINTER and CORE and 80 for CARDS into which the values given as parameters are edited according to the corresponding format phrase.

FORMAT PHRASES FOR WRITE

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameters
<u>Activate</u>	<u>Device implied or PRINTER</u>								
Aw.d or A(E1,E2)	Print 1 line	Skip w lines before printing	0	63	Skip d lines after printing	0	31		Non-editing does not require a parameter
	<u>Device CARDS</u>								
	Punch 1 card	ignored			ignored				
	<u>Device CORE</u>								
	Transfer as many characters from the print buffer into the string as the length of the st ng or print buffer allows	ignored			ignored				
<u>Boolean</u>	<u>Devices implied, PRINTER,CARDS,CORE</u>								
Bw or B(E1)	Place as many characters as possible of the strings TRUE or FALSE depending on the value of the parameter. Fill the rest of the field with blanks if necessary.	Field width (number of charac-ters used in the print buffer)	1	132 80 for CARDS	NOT ALLOWED			Left justified	BOOLEAN
<u>Decimal</u>	<u>Devices implied, PRINTER,CARDS,CORE</u>								
Dw.d or D(E1,E2)	Places the digits of a decimal number with d digits after the decimal point - leading zeroes suppressed, minus sign if negative.	Field	2	63	Provide d digits after decimal point	0	31	Right justified	INTEGER REAL REAL2 COMPLEX

FORMAT PHRASES FOR WRITE

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of paramete
<u>Eject</u> Ew or E(E1)	<u>Devices implied, PRINTER</u> Eject to logical line w-1. If the present position is past line w-1, ejection is to line w-1 on the next page. (Usually used to start at top of a page) <u>Devices CARDS, CORE</u> Ignored	Logical line number on page	1	72	NOT ALLOWED				Non-editing does not require a parameter
<u>Free</u> Fw or F(E1)	<u>Devices implied, PRINTER,CARDS,CORE</u> Read or write a field of w characters in free format. See sec. 8.6.1.	Field width	1	2047					INTEGER REAL BOOLEAN COMPLEX REAL2 STRING
<u>Integer</u> Iw.d or I(E1,E2)	<u>Device implied, PRINTER,CARDS,CORE</u> Place the digits of an integer number with minus sign if negative. The value is given to the base d. Where d=0 and d=10 have the same meaning.	Field width	1	63	Base for integer (e.g. octal use 8)	0	10	Right justified	INTEGER REAL COMPLEX REAL2 BOOLEAN (TRUE 1) (FALSE 0)

FORMAT PHRASES FOR WRITE

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameter
Absolute position to column Jw or J(E1)	Devices implied, PRINTER,CARDS,CORE The next phrase will start from column w.	Column number	1	132 80 for CARDS	NOT ALLOWED				Non-editing
Middle string Mw or M(E1)	Devices implied, PRINTER,CARDS,CORE The characters of the parameter are placed into the middle of the field. If the field width w is greater than the string length L then the string is preceded by (w-L)/2 blanks. If w is less than L then the right-most L-w characters of the parameter are lost.	Field width	1	132 80 for CARDS	NOT ALLOWED			Centre-justified	STRING
Left justified Integer Nw.d or N(E1,E2)	Devices implied, PRINTER,CARDS,CORE Same as I phrase exept that result is left justified.	Same as I phrase	1	63	Same as I phrase	0	10	Left justified	Same as I phrase
Real Rw.d or R(E1,E2)	Devices implied, PRINTER,CARDS,CORE Edits the parameter into the form ±X.XXX···X,±XX d significant digits Note: w > d+6	Field width (greater than d+6)	7	63	Number of significant digits	1	31	Right justified	INTEGER REAL REAL2 COMPLEX

FORMAT PHRASES FOR WRITE

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameter
String Sw or S(E1)	Devices implied, PRINTER, CARDS, CORE The characters of the parameter are placed into the field starting from the left. If the string length L exceeds the field width w then only the leftmost w characters are transferred, if w exceeds L then the rest of the field is blank.	Field width	1	132 80 for CARDS	NOT ALLOWED			Left justified	STRING
Real zero gives blanks Uw.d or U(E1,E2)	Devices implied, PRINTER, CARDS, CORE If value of the parameter is exactly zero then treat as Xw,	Field width	1	63	Ignored				
	otherwise treat as Dw.d	Field width	1	63	Provide d digits after the decimal point	0	31	Right justified	INTEGER REAL REAL2 COMPLEX
Integer zero gives blanks Vw or V(E1)	Devices implied, PRINTER, CARDS, CORE If value of the parameter is exactly zero then treat as Xw	Field width	1	63	Ignored				
	otherwise treat as Iw	Field width	1	63	Ignored			Right justified	INTEGER REAL REAL2 COMPLEX BOOLEAN

FORMAT PHRASES FOR WRITE

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameters
<u>Place blanks</u> Xw or X(E1)	<u>Devices implied, PRINTER,CARDS,CORE</u> Place w blanks into the print buffer	Number of blanks	1	132 80 for CARDS	NOT ALLOWED				Non-editing
<u>String Constant</u> String of characters enclosed in ' '	<u>Devices implied, PRINTER,CARDS,CORE</u> Place the characters in the number of columns required. Maximum length 132 for all devices but CARDS, which may have 80.								Non-editing

d) Actions when restrictions are broken

The following actions occur when any of the restrictions stated above are broken.

1. The print buffer at the error point is output on the appropriate device.
2. The message
   EDITING ERROR AT LINE XXXX. CHECK YOUR FORMAT
   is output on the PRINTER.
3. The corresponding parameter (if any) is bypassed.
4. Editing continues with the next parameter. The next field starts in the last column used by the phrase before the error occurred.

Common errors:

1. Parameter is of a type not allowed by the format phrase.
2. Field width is 0, too small to accept value, or too large.

e) Action when the end of the print buffer is reached

For devices implied, PRINTER or CORE ,if an editing phrase will cause editing beyond column 132 then the print buffer is output and editing begins again in column 1.

For device CARDS the limit is column 80.

f) Example showing differences between D, R and U phrases

```
BEGIN
  REAL X,Y,Z$
  FORMAT F(D12.4 ,R12.4,U12.4,A1)$
       X=Y=Z=3.14159&+1$
  WRITE (F,X,Y,Z)$
       X=Y=Z=0.0$
  WRITE (F,X,Y,Z)$
END$
```

Print lines

```
   31.4159      3.1416,+1        31.4159
          0            0
```

g) Example showing differences between I, N and V phrases

```
BEGIN
  INTEGER I,J,K$
  FORMAT F(I10,N10,V10,A1)$
       I=J=K=-31415$
  WRITE (F,I,J,K)$
       I=J=K=0$
  WRITE (F,I,J,K)$
END$
```

Print lines

```
      -3141531415      31415
           00
```

h) Example showing differences between M and S phrases

```
BEGIN
  STRING S(29)$
  FORMAT F(S40,A1,M40,A1)$
       S='THIS STRING HAS 29 CHARACTERS'$
  WRITE (F,S,S)$
END$
```

Print lines

```
    THIS STRING HAS 29 CHARACTERS
         THIS STRING HAS 29 CHARACTERS
```

8.6.5 Format phrases with READ

a) Use

Format phrases are used to inform the READ statement exactly where the characters making up the parameter can be found. There is also the special format F which allows Free Format to be used for a specified number of characters in the read buffer.

The read buffer is a string of 80 characters in length into which the contents of the card (for devices implied or CARDS) or of the string (device CORE) are placed for editing.

b) Form of a format phrase

Qw.d

or

Q(E1,E2)

where Q represents a formatting character (see below).

E1 must be an arithmetic expression with the same meaning as the integer constant w. The meaning and restrictions are given in section c.
E2 must be an arithmetic expression with the same meaning as the integer constant d. The meaning and restrictions are given below.

c) Available format phrases, meaning and restrictions with READ

The following table gives the possible format phrases and the restrictions attached to them.

FORMAT PHRASES FOR READ

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameters
<u>Activate</u> A	<u>Devices implied, CARDS</u> Transfer the contents of 1 card into the read buffer. Place the start for editing at the first character of the read buffer.	Ignored			Ignored				Non-editing
	<u>Device CORE</u> Transfer the contents of the string into the read buffer. If the string is greater than 80 characters transfer only the first 80 characters. If the string is less than 80 characters - say L characters, then the last 80 - L characters in the read buffer are unchanged. Place the start for editing at the start of the read buffer.	Ignored			Ignored				Non-editing
<u>Boolean</u> Bw or B(E1)	<u>Devices implied, CARDS,CORE</u> If the field contains anywhere in it the string TRUE or the character T or the integer constant 1 set the parameter to TRUE. For the string FALSE, character F or integer 0 set the parameter to FALSE Anything else in the field will cause an error.	Field width (number of columns reserved for the parameter)	1	80	NOT ALLOWED				BOOLEAN

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Allowed types of parameters
<u>Decimal</u> Dw.d or D(E1,E2)	<u>Devices implied, CARDS, CORE</u> Calculate a number from the digits in the field. Make it negative if preceded by a minus sign. The digits may have the form of an INTEGER, REAL or REAL2 constant as described in section 4.1. A comma (,) or the letter E may be used instead of & as the power of ten symbol.	Field width	1	63	If the number has no decimal point then place a decimal point before the digit which is d places to the left of the rightmost digit in the field else ignore	0	31	INTEGER REAL REAL2 COMPLEX
<u>Eject</u> Ew or E(E1)	Ignored by all devices							

FORMAT PHRASES FOR READ

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameters
Free Fw or F(E1)	Devices implied, CARDS, CORE Read the next w columns in the manner described in section 8.6.1. (Implied or free format)	Number of columns to be read in this way	1	80	NOT ALLOWED				INTEGER REAL BOOLEAN COMPLEX REAL2 STRING
Integer Iw or I(E1)	Devices implied, CARDS, CORE 1. Calculate a number from the digits in the field. Make it negative if a minus sign precedes. 2. Give the value a type according to the form of the number read (see section 4.1 for form of numbers). 3. Gonvert the number to integer. 4. Convert the result to the type of the parameter.	Field width	1	63	IGNORED				INTEGER REAL REAL2 COMPLEX
Position to column Jw or J(E1)	Devices implied, CARDS, CORE The next field to be edited starts in column w. (Useful for reread).	Column number of start of next field	1	80	NOT ALLOWED				Non-editing phrase

FORMAT PHRASES FOR READ

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Allowed types of parameters
<u>Middle String</u> Mw or S(E1)	Exactly the same as S.							
<u>Integer</u> Nw or N(E1)	Exactly the same as I.							
<u>Real</u> Rw.d or R(E1,E2)	Exactly the same as D.							
<u>String</u> Sw or S(E1)	<u>Devices implied, CARDS, CORE</u> Transfer as many characters as possible from the read buffer to the string given as parameter. Start with the leftmost character in the field into the leftmost character in the string. If the field is shorter than the string fill the rest of the string with blanks. If the string is shorter than the field then the rest of the characters in the field are lost. Note - a string quote is not taken as a string delimiter, but transferred like any other character.	Field width (Number of columns reserved for the string)	1	2047	NOT ALLOWED			STRING

FORMAT PHRASES FOR READ

Phrase	Action	w or E1 Meaning	w or E1 Min	w or E1 Max	d or E2 Meaning	d or E2 Min	d or E2 Max	Position in field	Allowed types of parameters
<u>No change if blanks real</u> Uw.d or U(E1,E2)	<u>Devices implied, CARDS, CORE</u> 1. If the field reserved is completely blank treat as Xw. 2. Otherwise treat as Dw.d.	Field width Field width	1 1	63 63	Ignored Same as for D				INTEGER REAL REAL2 COMPLEX
<u>No change if blanks integer</u> Vw or V(E1)	<u>Devices implied, CARDS, CORE</u> 1. If the field reserved is completely blank treat as Xw. 2. Otherwise treat as Iw.	Field width Field width	1 1	63 63	Ignored Ignored				INTEGER REAL REAL2 COMPLEX
<u>Blanks</u> Xw or X(E1)	<u>Devices implied, CARDS, CORE</u> Skip the next field of w columns.	Field width	1	80	NOT ALLOWED				Non-editing
<u>String constant</u> String of charafters enclosed by ' '	Completely ignored								Non-editing

d) Action when restrictions are broken

The follwing actions occur when any of the restrictions given above are broken.

1. If an error label is present (the third label of the label list), a jump is made to that label.

2. If no error label is present, the read buffer is printed on the printer and a marker is printed showing the exact position where the error occured and the line number of the program being executed.

Common errors:

1. Parameter is of a type not allowed by the format phrase.

2. Restrictions on w or d have been broken.

3. The characters in the field specified are illegal or do not have the correct form. (For example spaces are not allowed in a numeric constant).

8.3.6 Repeat Phrases

a) Definite repeats

Use

Instead of writing out the same format phrase or group of phrases several times, it is possible to specify the number of times the phrase or phrases should be referred to by using a repeat phrase.

Form

```
 nQw.d
 n(Qw.d,Qw.d,.......Qw.d)
:E:(Qw.d)
:E:(nQw.d,:E:(Qw.d),:E:(nQw.d))
etc.
```

where n is a positive integer constant
Q is any format phrase (editing or non-editing)
E must be an arithmetic or boolean expression
w and d have the meanings given in section 8.6.4. and 8.6.5.

Rules

i) The expression E is evaluated when the repeat phrase is activated. That is when the format phrase is required, before the parameter is evaluated.

ii) If E > 0 the format phrase (S) are repeated that many times. If E = TRUE the phrases are taken once.

iii) If E $\leq$ 0 or E = FALSE the format phrase(s) which this repeat controls, will be skipped.

iv If E has an integer value greater than 2047, an error will occur.

Examples:

```
BEGIN
  COMMENT PRINT AN ARRAY WITH ONE COLUMN PER LINE$
  INTEGER N,M$
  ARRAY X(1:N,1:M)$
  FORMAT F6(:M:(:N:(R16.8),A1))$
  WRITE (F6,X)$
END$
```

b) Indefinite Repeats

Use

It is possible to repeat certain groups of format phrases an indefinite number of times depending only on the number of elements in the input/output list.

Form

The groups of phrases to be repeated are enclosed in parentheses without a repeat expression preceding. The delimiters <<   >> of an inline format and the outermost brackets of a declared format also denote indefinite repeat.

Warnings

i) Indefinite repeat groups should in most cases have an activate (A) phrase in them since all format phrases beyond the group are ignored. If they do not, a warning message is given.

ii) Errors can occur when two cards are read instead of one because the input list is longer than the number of phrases in the format.

iii) Attempts to cause an indefinite repeat of a format containing only non-editing phrases will cause the format to be cancelled.

Examples:

```
BEGIN
  COMPLEX ARRAY COMPARRAY (1:50,1:50)$
  INTEGER SIZE,I$
  FORMAT FREAD(A,I12,(A,10R8.2))
         FWRITE('COMPARRAY OF SIZE',I12
         A1.2,(10(R9.2,X2),A1))$
  READ (CARDS,FREAD,SIZE,FOR I=(1,1,SIZE)
        DO FOR J=(1,1,SIZE) DO COMPARRAY A(I,J))$
  COMMENT WILL READ IN THE PART OF THE ARRAY REQUIRED$
  WRITE (PRINTER,FWRITE,FOR I=(1,1,SIZE)
        DO FOR J=(1,1,SIZE) DO COMPARRAY(I,J))$
  COMMENT WILL PRINT OUT HEADING AND THEN THE PART
        OF THE ARRAY REQUIRED$
END$
```

```
BEGIN
  INTEGER I$
  COMPLEX C$
  FORMAT FREAD(A,I12,R12.6)$
  READ (CARDS,FREAD,I,C)$
  COMMENT WILL READ TWO CARDS SINCE COMPLEX VALUES
                                REQUIRE TWO PHRASES$
END$
```

8.7 Input/Output List

Use

The input list is an ordered set of variables into which values can be transferred. The output list is an ordered set of expressions which can be evaluated and their values transferred to the required output device.

Form

The list may have two forms

Declared list
Inline list

8.7.1 Inline List

Use

To give the input or output statement a list of expressions to or from which values may be transferred.

Form

Any ordered group of expressions which are parameters to an input or output procedure is an inline list.

Examples:

```
       FORMAT F(A,3R12.2)$
       REAL X,Y,Z,A,B,C$
       WRITE (X,Y,Z)$
       READ (CARDS,F,EOFLB,A,B,C)$
EOFLB: COMMENT THE EXPRESSIONS X,Y,Z,A,B,C, IF A GTR B THEN
       A-B ELSE B-A, ARE ALL MEMBERS OF INLINE LISTS$
```

8.7.2 Declared List

Use

When several input or output calls require the same expressions in the same order a declared list may be used.

Form

LIST <identifier>(<list elements>)$
It must obey the rules for declarations.
Several lists may use one declaration.

Examples:

```
LIST L1(FOR I=(1,1,5) DO A(I),X,Y),
     L2(IF B THEN X ELSE Y,Z)$
```

8.7.3 Rules for Lists

a) Arrays

i) An array identifier may be used without subscripts in a list.
The meaning of this is that every element in the array is to be used in the list.

ii) For multi-dimensional arrays, the left most subscript varies most frequently.

Example:

```
ARRAY X(1:2,1:3,1:4)$
WRITE (CARDS,X)$
COMMENT WILL PUNCH OUT THE ELEMENTS IN THE FOLLOWING
                                                ORDER
X(1,1,1), X(2,1,1), X(1,2,1), X(2,2,1),
X(1,3,1), X(2,3,1), X(1,1,2), X(2,1,2),
X(1,2,2), X(2,2,2), X(1,3,2), X(2,3,2),
X(1,1,3), X(2,1,3), X(1,2,3), X(2,2,3),
X(1,3,3), X(2,3,3), X(1,1,4), X(2,1,4),
X(1,2,4), X(2,2,4), X(1,3,4), X(2,3,4)
```

b) Other expressions

The expression is evaluated at the time the list element is referenced.

c) Format in lists

A format identifier or inline format may be placed in a declared list.

d) List with MAX and MIN

The parameters to MAX and MIN may be given in a declared list.

8.7.4 Sublists

Use

Lists or list elements may be grouped so that they can be repeated in a specific order.

Form

Sublists are formed by enclosing the list elements with brackets.

Example:

```
LIST L1(FOR I=(1,1,2) DO (A(I),B(I)))$
```

Note:

List elements are expressions and therefore cannot be enclosed within BEGIN END. Sublists must be used whenever such a construction is required.

8.8 Input/Output Statements

8.8.1 The READ Statement

Use

To specify that values are to be input according to the given parameters.

Form

```
READ(<device>,<format list>,<input list>,<label list>)$
```

Devices allowed

The allowed devices are implied, CARDS, CORE, TAPE, DRUM.

Labels

Up to 3 labels may be used. See Sec. 8.5.

8.8.2 The WRITE Statement

Use

To specify that values are to be output according to the given parameters.

Form

```
WRITE (<device>,<format list>,<modifier list>,<output list>,
       <label list>)$
```

Devices allowed

The allowed devices are implied, PRINTER, CARDS, CORE, TAPE, DRUM.

Example:

```
WRITE(TAPE('A'),ERRLB,EOF('XYZ'),X,Y,Z)$
WRITE(CORE(S),<<3R12.2,A>>,X,Y,Z)$
```

Labels

Only 1 label is allowed. See sec. 8.5.

8.8.3 The POSITION Statement

Use

To position a specified magnetic tape unit or sequential drum file to a position specified by a modifier.

Form

```
POSITION(TAPE(<parameter>),<modifier list>,<integer expression>,
              <label list>)$
```

Devices allowed

Only TAPE is allowed as a device.

Labels

Up to 3 labels may be used. See sec. 8.4, 8.5.

Position by number of records

The integer expression specifies the number of records to be positioned. If it is positive, the positioning is done in the forward direction, if negative in a backwards direction. If the device is a sequential drum file, only positioning forward is allowed.

8.8.4 The REWIND and REWINT Statements

Use

REWIND positions a magnetic tape or sequential drum file to its starting position.
REWINT rewinds a magnetic tape and locks it so that it can no longer be used, or rewinds a sequential drum file to its start position.

Form

```
REWIND(TAPE(<parameter>))$
REWINT(TAPE(<parameter>))$
```

Device allowed

Only device TAPE is allowed with these operations.
Any other devices will cause undetectable errors.

8.8.5 The MARGIN Statement

Use

To change the margin settings on the printer. Depending on the size of paper used at an installation, there will be a certain number of lines per print page.
Procedure MARGIN allows the user to specify which is to be the first line and which is to be the last line on page.

It can also be used when special print forms such as labels or envelopes are being printed.

Form

```
MARGIN (<length>,<top line number>,
        <bottom line number>,
        <string if desired>)$
```

Where

<length> is an integer expression specifying the number of lines the installation allows per page.

<top line number> is an integer expression specifying the logical line number where the first line is to be printed.

<bottom line number> is an integer expression specifying the logical number where the last line is to be printed.

<string> is a string which is typed on the console when margins are actually changed on the printer.

Example:

```
BEGIN
  BOOLEAN B$
MARGIN (IF B THEN 72 ELSE 66,5,
        IF B THEN 69 ELSE 63)$
END$
```

9 OTHER INFORMATION

9.1 Comments.

Use

The use of explanatory messages is encouraged to aid readability of the program and to help in finding errors in the source text.

Methods.

a) After BEGIN or any $ or ; the following construction may be placed.

COMMENT any characters not including ; or $ followed by ; or $

b) After END comments can be placed.
However, the characters ; or $ or the words END or ELSE cause the ending of the comment.

c) In a procedure declaration comments may be placed in the formal parameter list by substituting for the comma the construction:

)<letter string>:(

(See section 7).

Example:

```
COMMENT THIS PROGRAM SHOWS COMMENTS$
BEGIN COMMENT CAN COME AFTER BEGIN$
  INTEGER I$
COMMENT CAN COME AFTER DECLARATION$
PROCEDURE SHOW (K) WORDS CAN BE PLACED HERE: (L)$
  REAL K,L$
    K=L$ COMMENT CAN COME AFTER A STATEMENT$
    IF I GTR 50 THEN
BEGIN
  SHOW (I,50-I)$
END YOU CAN ALSO PUT COMMENTS HERE
```

```
ELSE
  SHOW (I,50-I )$
END OF THIS PROGRAM SHOWING COMMENTS$
```

Note:

A comment may come before the first BEGIN of a program.

9.2 Options

It is possible to control certain actions of the ALGOL compiler and run-time system by placing a specific option letters after the masterspace on the ALGOL processor card or the XQT card. (See EXEC II manual page 3-1).

At compiletime these same options may also be turned on by using a "statement" of the form

```
OPTION 'string of option letters'$
```

They may be turned off by using

```
OPTION 'string of option letters' OFF$
```

These "statements" are accepted wherever declarations or statements are allowed.

Note:

OPTION may come before the first BEGIN.

Available options on the ALG-card.

A   Accept the compiled program even if errors are found. No warning messages are given.

E   All external procedures when they are compiled require this option.

F   The compiled SLEUTH II code is listed and punched into cards, which are accepted by the SLEUTH II assembler.

G   The listing for this compilation will start at the top of a new page.

L The SLEUTH II code produced by the compiler will be listed. The instructions resulting from each line of ALGOL text will appear just before the line is printed.

N The source text listing is suppressed. No warnings are given, but error messages are printed together with the source lines to which they apply.

O This option has the same effect as R.

R This option removes the instructions which check wheter the subscript being used is within the bounds declared for the array.
It is suggested that this option should <u>not</u> be used during debugging. Production programs can benefit greatly from the saving in time when the check is removed.

S Punch the updated symbolic text in compressed form.

T At the end of the listing, times are given for the four passes of the compiler and the total time taken for the compilation. The number of words used on drum for the intermediate output from the passes of the compiler is also printed.

V Suppress warning messages.

W Correction cards used to update a symbolic version are listed before the normal source text listing.

X If errors are detected in the compilation, the entire run is aborted.

Z No run-time diagnostic information is prepared. When this option is used, a PMD card may not be used. The program will <u>not</u> keep track of the line numbers being executed so that run-time error message will not be complete. The use of this option saves time and core-space in production programs, but should not be used when debugging.

Available options on the XQT card

(See also EXEC II manual sec. 5.8).

A Accept the program for execution even though errors have been found during compilation or allocation. If compile-time errors have occured, execution will proceed up to the point of the first error and then the program is terminated with the message:

SOURCE LANGUAGE ERROR AT LINE XXX

F This option must be used when using external FORTRAN, procedures containing double precision or complex arithmetic. Otherwise the program will terminate with the message'

ILLEGAL OPERATION AT LINE XXX

where the line number refers to the last ALGOL line executed.

N Suppress listing of allocation tables.

X Abort the rest of the run if errors occur.

9.3 Chained Programs and NU ALGOL

1. The EXEC II manual Section VI.2. describes how large programs may be broken into sections or links. NU ALGOL programs may also take advantage of this feature through the use of the statement

   CHAIN (<integer expression>)$

   where the value of the <integer expression> is the number of the next link to be executed.

2. Sequential drum files may be used across links because Y$TTAB, their control table, is kept in blank common.

3. Device DRUM may be used across links. The current drum position, obtained by the procedure DRUMPOS, is not destroyed.

4. No data from the ALGOL programs is saved across links because no data is kept in blank common.

5. Users of external FORTRAN or SLEUTH programs which have blank common, must ensure that their data areas do not interfere with Y$TTAB.

10 ERROR MESSAGES

Security

The compiler tries to catch and properly diagnose all errors in the text given to it. Sometimes the syntax is so incorrect that it confuses the compiler to the point where spurious messages are printed or certain internal errors may occur. When such internal errors occur it is suggested that all other errors diagnosed be corrected. In most cases, the internal error will then disappear.

Diagnostics.

Where possible the exact syntax causing the error is marked with an asterisk. The following list suggests the possible problem and if possible gives a reference to where the required rules are explained. The user's help in suggesting other possible problems detected and diagnosed under specific error messages will be appreciated.

Level of errors.

There are three levels of errors.

a) Warnings - are given when a construction may cause an error if not used correctly, or the construction is inefficient They are not counted in the total given in the line

   XX ERROR(S) WERE FOUND

   They can be suppressed by using the V option or as a side-effect of the A or N options.

b) Errors - These are the usual diagnostics given when the compiler cannot translate the given source code into meaningful object code.

   The program produced by the compilation may be loaded and executed by using an A option on the XQT card but when a statement containing an error is executed, a jump will be made to a run-time error routine which terminates the program.

c) Compilation killers - For certain internal compiler errors or table overflows and such unresolvable problems as IMPROPER BLOCK STRUCTURE, compilation is immediately stopped. Not all errors are detected. In these cases an XQT card even with an A option will do nothing because no program has been produced.

10.1. Compile-Time Error Messages

Error number	Message	Possible problem
1.	Illegal number	The number does not conform to the syntax of sec.
2.	Illegal character	Some special characters cannot be used outside strings or comments. (See section 2.1).
3.	Correction card error	Line number on correction cards are not in ascending order.(See EXEC II· 5-10A)
4.	Improper use of reserved identifier	Reserved identifiers (see section 2.2.) may only be used with their special meaning.
5.	Too long string	String constants may not have more than 132 characters. A string quote may be missing or an extra one has been punched.
6.	Missing delimiter	Missing operator such as + or - or missing $ on previous statement.
7.	Wrong delimiter	The compiler is expecting some other delimiter. Also VALUE must come before all specifications.
8.	Improper operand, or operand is missing	Usually two operators have been placed together. For example A×-B is not allowed. A×(-B) must be used.
9.	Missing operand	Improper construction of an IF statement. (See section 5.4).
10.	Illegal construction	Often caused by a mismatched number of left and right parentheses or any other non-standard construction.
11.	Missing specification of <name of variable>	No specification given for a parameter to a procedure. (See section 6.1)
12.	Pass 1 stack overflow	An internal compiler error usually caused by other errors or a too large program.

Error number	Message	Possible problem
15.	Double pecification of <name of variable>	A parameter to a procedure has been specified twice. (See section 7.1)
16.	Illegal value specification of <name of variable >	LABEL , LIST , FORMAT , SWITCH and PROCEDURE cannot be given a value specification.
17.	Missing formal parameter	A specification has been given for a variable which is not a parameter to the procedure. Often it should be a declaration of a local variable and come inside the BEGIN of the procedure.
18.	*Warning* Improper termination - remaining cards ignored	All BEGIN's have been matched with END's but still some cards remain.
19.	*Warning* Missing end - extra end interested	The block structure may not be quite correct or the final END has been forgotten.
20.	Too many nested BEGIN-END pairs	Only 34 nested BEGIN-END pairs or 9 block levels are permitted.
21.	Improper block structure	Some BEGIN's or END's missing, possibly caused by other errors.
22.	Too many errors-compilation suppressed	Have you read the programmer's guide?
23.	Double declaration of <name of variable> at line <line of second declaration>	Two identifiers in which the first twelve or less characters are the same, have been declared in the same block.
24.	Missing declaration of <name of variable>	An identifier has been misspelled or the user has forgotten to declare it.
25.	Redeclaration stack overflow	There are too many identifiers with similar spellings in nested blocks.

Error number	Message	Possible problem
26.	Interphase 1 rror	An internal compiler error. Check for other serious errors.
27.	Internal error	The user has totally confused the compiler. Correct all other errors and try again.
29.	Accumulator stack overflow (simplify this expression)	There are too many intermediate results in an arithmetic expression for the computer to handle.
30.	Mixed types in left part list.	In multiple assignments all variables must have the same type.
31.	Illegal (after <name of variable> at line <line of declaration>	Possibly a delimiter is missing or a simple variable is being used with a subscript.
32.	Wrong number of subscripts to array	The number of subscripts used must always match the number of dimensions given for an array in the declaration.
33.	Improper type in expression	Only certain transfer functions exist between different variable types. This expression requires one which does not exist. (See section 7.4.).
34.	Wrong parameter kind to procedure <procedure name> at line <line of declaration>	Formal and actual parameter kinds must match. For example the actual parameter may not be an array identifier when the formal one is a simple variable. (Line 0 refers to a standard procedure.)
35.	Wrong parameter type to procedure <procedure name> at line <line of declaration>	The type of an actual parameter must match that of its formal parameter unless a transfer function exists. Note that no transfer functions are allowed for arrays. (Line 0 refers to a standard procedure.)

Error number	Message	Possible problem
36.	Illegal assignment	A transfer function which does not exist has been called for.
37.	Constant table overflow	The program contains a constant expression which is too complicated, or the total number of constants in the program is too large
38.	Wrong number of parameters to procedure <procedure name> at line <line of declaration>	The number of parameters is a procedure call does not match the declaration. (Line 0 refers to a standard procedure.)
39.	Improper type in bound pair list of array <array name>	Only INTEGER, REAL and REAL2 are allowed types for substrict bounds in array declarations.
40.	*Warning* Do you want to compare constants?	Possible puncing error
41.	Improper type before THEN	Only boolean expressions are allowed before the delimiter THEN.
42.	Improper relation between complex or string expressions	Strings and complex numbers can only be compared for equality or non-equality.
43.	Undefined transfer function	An implicit non-existent conversion has been called for. (See section 7.4.).
44.	Operand stack overflow	Internal compiler error. Check carefully for other errors. The program is too complicated.
45.	Improper type of controlled variable <name of variable> at line <line of declaration>	The controlled variable in a FOR loop may only be of type INTEGER or REAL.
46.	*Warning* Zero step	The controlled variable will not be changed in a FOR statement when the step is zero.

Error number	Message	Possible problem
47.	Improper type in FOR list element	Only INTEGER and REAL types are allowed in a FOR list.
48.	Wrong type of subscript for array <array name>	Only INTEGER, REAL and REAL2 are legal types for subscripts.
49.	Operator stack overflow	Internal compiler error. Check carefully for other errors. The program is too large and complicated.
50.	FOR stack overflow	Only 24 nested FOR statements are allowed or a FOR-list may contain about 40 elements.
51.	*Warning* Reference into FOR-statement by label <label name> at line <line of declaration>	Jumps to labels in FOR-statements are hazardous since the loop control may not be initialized correctly.
52.	*Warning* Test for equality between non-integers may be meaningless	Variables of types REAL, REAL2 and COMPLEX are only approximations to a value and hence may not be exactly equal.
53.	Too many different identifiers	Approximately 600 different identifiers may be used.
54.	Pass2 stack overflow	Internal compiler error. Check for other errors which may have caused the compiler confusion. The program may have too many declarations.
55.	Unrecoverable error in ALGOL drum file	Internal compiler error. Check for other errors which may have confused the compiler - or for a machine failure.
56.	Overflow in ALGOL drum files-program too large	The intermediate outputs from the compiler are larger that the scratch area on drum.
57.	Improper format construction	Some rule for formats has been broken (See section 8.6).
58.	Zero replicator	Although replicator expressions may have the value zero, the constant replicator zero has no meaning.

Error number	Message	Possible problem
59.	Missing right or extra left parenthesis	The number of right and left parentheses used in a format do not match
60.	Missing left or extra right parenthesis	The number of right and left parentheses used in a format do not match
61.	Improper field specification	The field width part of a format phrase (w) is not formed properly. (See section 8.6).
62.	xWarningx Missing activate within indefinite repeat	Indefinite repeat formats usually require an A-phrase to perform properly.
63.	xWarningx Specified field is longer than one line	The field width part of a format phrase (w) has little meaning if it exceeds 132 columns.
64.	Format stack overflow	Only 10 sets of nested brackets are allowed in a format.
65.	xWarningx Timeconsuming conversion to integer subscript in array <array name>	It is allowable to use non-integer expressions for subscripts, but it is very slow.
66.	Illegal format character	Only certain characters are meaningful within a format. (See section 8.6.).
67.	This feature is not implemented	The construction cannot yet be compiled.
68.	Unrecoverable error in source input files	Trouble with reading symbolic version of program from the card reader, tape or PCF area on drum. Usually a hardware error.
69.	Interphase 2 error	Internal compiler error - check for other possible errors.
70.	Pass 1 stack underflow	Internal compiler error - check for other possible errors.
71.	Operandstack underflow	Internal compiler error - check for other possible errors.

Error number	Message	Possible problems
72.	Improper use of formal parameter <parameter name> at line <line of specification>	A formal parameter not specified as a procedure is being used like a procedure. Example: PROCEDURE P(X); REAL X; BEGIN X; END;
73.	Conversion to integer causes overflow	REAL and REAL2 constants may have a largest absolute value of about 10^{38} but integer constants have a largest absolute value of only about 10^{11}.
74.	Improper parameter to string <string name>	The parameters to a string may only be INTEGER, REAL or REAL2 expressions.
75.	Too many parameters to string <string name>	Strings require either no parameters or only a starting character position and the length. (See section 4.4).
76.	Operator stack underflow	Internal compiler error - check for other possible errors which could have confused the compiler.
77.	xWarningx Inconsistent use of dimensions to array <array name>	A formal array has been used with different numbers of subscripts.
78.	Parameter out of range in procedure procedure <procedure name>	Certain standard procedures require parameters to have value in a certain range.
79.	Missing BEGIN	All programs except externally compiled procedures must start with BEGIN. It is not allowed to place a label before the first BEGIN.
80.	xWarningx Operand for // is not integer	Integer divide (//) is only allowed for integers. Conversion will be attempted. This warning is given to the rules for ALGOL 60.

Error number	Message	Possible problem
81.	Division by zero	Division by zero has been attempted in a constant expression being avaluated by the compiler.
82.	Too many string constants	There may be at most 200 string constants in a program except for the ones used in formats.
83.	Too many labels	A program may contain 200 label declarations.
84.	Too many external references	A program may reference 50 external procedures including standard procedures and system subroutines.
85.	Too many procedure parameters	A procedure may have up to 63 parameters. For LIBRARY procedures the number is determined as shown in sec. 7.3.5.2.
86.	Prototype table overflow	The program contains to many and too large blocks or procedures.
87.	Too many external procedures	Only 10 external procedures may be compiled within the same element.
88.	Too many array and string declarations	The program has too many arrays or string with different bounds.

10.2 Run-Time Error Messages

Because the evaluation of many expressions is left to the run-time routines, certain errors can occur. These are caught by the run-time system and the appropriate messages given, together with the line number of the element where the error occured.

Number	Message	Possible problem
0.	Internal error	Trouble in an ALGOL run-time routine Consult your systems support people.
1.	Improper type conversion	A transfer function which is not allowed has been requested.
2.	This feature is not implemented	The run-time routines of the compiler cannot process this construction.
3.	Incorrect number of parameters	The number of parameters in the procedure call does not match the number given in the procedure declaration.
4.	An attempt has been made to store into a constant	A formal parameter appearing to the left of an assignment has a constant as its actual parameter. There may be a missing value specification or the parameters in the procedure call may not be in the correct order.
5.	An attempt has been made to store into an expression	A formal parameter appearing to the left of an assignment has an expression as its actual patameter. Perhaps the parameters in the procedure call are not in the same order as those in the procedure declaration, or a value specification is missing.
6.	Number too large	A REAL, REAL2 or the real or imaginary parts of a COMPLEX number having absolute value larger than about 10^{38} has been produced.
7.	Attempted division by zero	The divisor in an integer or real division is zero.

Number	Message	Possible problem
8.	Store error	Incorrect code generated by compiler due to errors in the source code, program destroyed by FORTRAN or machine language procedures, or subscript out of range when using R-option.
9.	Illegal operation	Missing external procedure or incorrect return from a FORTRAN or machine language procedure.
10.	Result undefined for conversion	The result produced by a transfer function is not a meaningful value.
11.	MERR$ termination	Execution of the run has been terminated by the system error exit routine. (Often maximum time or pages.)
12.	Memory capacity exceeded	Usually caused by array bounds which are too big, or by the dynamic creation of too many or too large procedures.
13.	Improper type of parameter	The type of an actual parameter must match that of its formal parameter unless a transfer function exists. Note - no transfer functions are allowed for arrays.
14.	Improper kind of parameter	Formal and actual parameter kinds must match. For example the actual parameter may not be an array identifier when the formal one is a simple variable.
15.	Argument out of range	A parameter to a standard procedure is not within the limits accepted by that procedure.
16.	Subscript out of range	The subscript computed for an array element does not fall within the bounds specified in the array declaration.
17.	Too many dimensions	Only 10 dimensions are allowed in an array.

Number	Message	Possible Cause
18.	Read error	Problem with using the READ statement, usually because of an undefined transfer function or a constant not in the correct format.
19.	Improper array bound in declaration	The evaluation of the expressions in an array bound has prodused a lower bound that is greater than the upper bound.
20.	Blocklevel is too high - no more X registers	Only 9 nested block levels are allowed.
21.	A control card was read by the read statement	If not done for a reason, this message usually implies that the amount of input data is known in correctly. Sometimes when reading cards, it is caused by reading two or more cards instead of one because of an incorrect FORMAT or LIST, or because free format READ always starts on a new card.
22.	Improper parameter	Improper parameter in size or sign.
23.	Attempt to read/write beyond random drum limits	The parameter to device DRUM is too large or is negative.
24.	Input/output error	Error with device DRUM or TAPE. Often caused when the length of an input list is not the same as that of the corresponding output list.
25.	Source language error	Executions done with A-option can only procede as far as the first error.
26.	Improper type of controlled variable	The controlled variable of a FOR statement is a formal parameter and

Number	Message	Possible Cause
		the corresponding actual parameter is not of the same type.
27.	Write error	Improper parameters given to the WRITE statement.
28.	Zero or negative string length in declaration	The expression given as the length of the string has a value less than 1.
29.	Checksum error	The checksum on a tape record is not correct. Possible tape error or incompatible tape format.
30.	Tape error	Beyond end of information if sequential drum file, or actual tape error and no error label avaiable.
31.	Too many labels	WRITE may only have 1 label. READ and POSITION may have 3 labels.
32.	Position error	Improper parameters given to the POSITION statement or trouble in positioning a file.
33.	List longer than record	The input list given to READ with device TAPE is longer than the record on tape.
34.	Formats are not allowed with TAPE or DRUM	Devices TAPE and DRUM may not read or write formatted data.
36.	Only ten nested sets of parentheses allowed.	In a format there can only be 10 nested sets of parentheses.

Number	Message	Possible Cause
37.	Neither labels nor lists allowed in lists.	The list elements for a declared list can only be expressions, array identifiers or formats.
38.	Input or format error in READ	The form of an item being read and the format used are not compatible. The input image is printed with an asterisk showing where the error occurred.
39.	Editing error in WRITE. Check your format	The value to be edited is too large for, or in some other way incompatible with the format. The output buffer is printed showing how far the editing has progressed. The editing will continue with the next value.

APPENDIX A.

BASIC SYMBOLS

Out of the 64-character set of the UNIVAC 1107/1108 computers, 55 characters are recognized by the NU ALGOL compiler as being meaningsful within an ALGOL program. (See sec. 2.1). The remaining 9 characters have no interent meaning and are allowed only within strings. They may thus be installation defined.

To the compiler the meaning of a character is determined by the value of its internal representation ("field data" value). The table below lists the characters by their internal representation together with a common graphic representation. The corresponding punched-card codes are not shown because they may be installation defined. For the installation defined characters no graphic is shown.

Table I. NU ALGOL characters

Internal value (octal)	Graphic symbol	Internal value (octal)	Graphic symbol	Internal value (octal)	Graphic symbol
00		25	P	52	
01	[	26	Q	53	:
02	]	27	R	54	
03		30	S	55	
04		31	T	56	,
05	SPACE	32	U	57	
06	A	33	V	60	0
07	B	34	W	61	1
10	C	35	X	62	2
11	D	36	Y	63	3
12	E	37	Z	64	4
13	F	40	)	65	5
14	G	41	-	66	6
15	H	42	+	67	7
16	I	43	<	70	8
17	J	44	=	71	9
20	K	45	>	72	'
21	L	46	&	73	;
22	M	47	$	74	/
23	N	50	*	75	.
24	O	51	(	76	
				77	

The basic symbols of the NU ALGOL hardware language are represented by means of the above characters. The following table shows these symbols along with the corresponding symbols of the ALGOL 60 reference language.

Table II. NU ALGOL Basic Symbols

ALGOL 60	NU ALGOL	ALGOL 60	NU ALGOL
true	TRUE	;	; or $
false	FALSE	:=	= or :=
+	+	step	STEP
-	-	until	UNTIL
×	*	while	WHILE
/	/	comment	COMMENT
÷	//	(	(
↑	**	)	)
<	LSS	[	(or [
≤	LEQ	]	) or]
=	EQL	‘	'
≥	GEQ	’	'
>	GTR	begin	BEGIN
≠	NEQ	end	END
≡	EQIV	own	
⊃	IMPL	boolean	BOOLEAN
∨	OR	integer	INTEGER
	XOR	real	REAL
∧	AND		REAL2
¬	NOT		COMPLEX
go to	GO TO		STRING
	or GOTO or GO	array	ARRAY
if	IF	switch	SWITCH
then	THEN		FORMAT
else	ELSE		LIST
for	FOR		LOCAL
do	DO		EXTERNAL
	OPTION		ALGOL
	OFF		FORTRAN
,	,		LIBRARY
.	.		SLEUTH
$_{10}$	& or &&	procedure	PROCEDURE
:	: or ..	label	LABEL
		value	VALUE

APPENDIX B.

EXAMPLES OF PROGRAMS

This appendix contains some simple examples illustrating the use of NU ALGOL
Each has been run on the 1108 and some sample input and results are shown.

```
BEGIN
COMMENT                        EXAMPLE 1
         CALCULATION OF VALUE OF ARITHMETIC EXPRESSION
         WITH READ IN VARIABLES $
REAL     A,B,C $
INTEGER  TOILL $
   READ (CARDS,A,B,C) $
   TOILL = A+B**C/A $
   WRITE (PRINTER,A,B,C,TOILL) $
```

DATA

```
5 6.2 1.222
```

RESULTS:

```
5.0000,+00   6.2000,+00   1.2220,+00    7
```

```
BEGIN
COMMENT                        EXAMPLE 2
         CALCULATION OF SQUAREROOT, B, OF A REAL NUMBER,
         A, WITH 6 DIGITS ACCURACY BY NEWTON-RAPHSON ITERATION $
REAL     A,B,OLDB $
   READ (CARDS,A) $
   OLDB = 1.0 $
   FOR B = 0.5*(A/OLDB+OLDB) WHILE ABS(B-OLDB) GTR 10**(-6)*B DO
      OLDB = B $
   WRITE (PRINTER,A,B) $
END PROGRAM $
```

DATA

```
5.77777
```

RESULTS:

```
5.7778,+00 2.4037,+00
```

```
BEGIN
COMMENT                          EXAMPLE 3
          VALUE OF A POLYNOMIAL Y=B(0)+B(1)*X....... +B(N)*X**N  $
REAL      X,Y $
INTEGER   K,N $
   READ (CARDS,N) $
COMMENT   DEGREE OF POLYNOMIAL READ FROM CARDS. INNER BLOCK PERFORMS
          READING OF COEFFICIENTS AND CALCULATIONS AND PRINTING OF
          RESULTS $
   BEGIN
   REAL ARRAY B(0:N) $
      READ (CARDS,B) $
      READ (CARDS,X) $
      Y = B(N) $
      FOR K=N-1 STEP -1 UNTIL 0 DO Y = Y*X+B(K) $
      WRITE (PRINTER,'VALUE OF A POLYNOMIAL OF DEGREE','N=',N,
            'COEFFICIENTS',B,'X=',X,'Y=',Y) $
   END CALCULATION $
END PROGRAM $
```

DATA

```
4
1.223 3.5 7.52 -4.02 -33.5
5.55
```

RESULTS:

```
VALUE OF A POLYNOMIAL OF DEGREE
N=
           4
```

COEFFICIENTS

```
  1.2230,+00  3.5000,+00  7.5200,+00 -4.0200,+00 -3.3500,+01
X=
  5.5500,+00
Y=
 -3.2220,+04
```

```
BEGIN
COMMENT                    EXAMPLE 4
         PROGRAM WITH A REAL PROCEDURE, BIG, WHICH FINDS THE LARGEST
         OF THE N LOWER-INDEXED ELEMENTS (STARTING WITH INDEX=1) OF A
         ONE-DIMENSIONAL ARRAY, A, WITH POSITIVE ELEMENTS $
   REAL PROCEDURE BIG(N,A) $
   VALUE N $
   INTEGER N $
   REAL ARRAY A $
   BEGIN
     INTEGER B $
     REAL  C,D $
     B = 1 $
     D = A(1) $
L:   C = D - A(B+1) $
     IF C LSS 0 THEN D = A(B+1) $
     B = B+1 $
     IF B LSS N THEN GO TO L $
     BIG = D $
   END BIG $
REAL ARRAY F(1:50) $
REAL H,K $
 READ (CARDS,F) $
COMMENT  CALL OF BIG TO FIND THE LARGEST OF THE 20 LOWER
ELEMENTS OF F $ H = BIG(20,F) $
 WRITE (PRINTER,H) $
COMMENT  LARGEST ELEMENT IN F $
 K = BIG(50,F) $
COMMENT  USE OF BIG IN MORE COMPLEX EXPRESSION $
 H = H + BIG(10,F)/K*BIG(15,F) $
 WRITE (PRINTER,H,K) $
END PROGRAM $
```

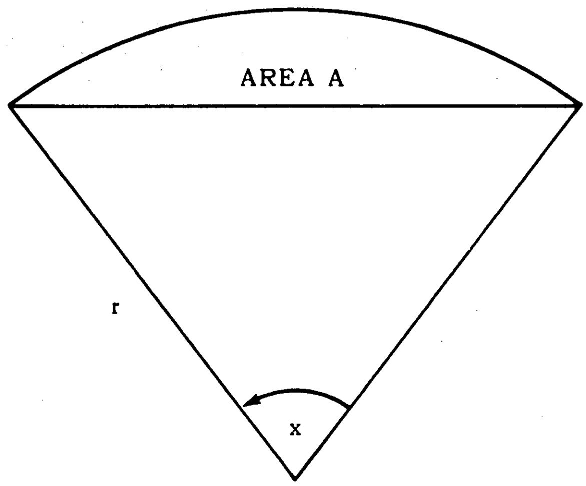
DATA

```
1.22 3.55 1 22.2 0.5 7.2 8.12 21.4 4.1 22.5 0.422
55.2 0.12345 5.88 3.55 7.53 4 5 2 3 1 77 5 22.1
5.1 2.3 3.2 4.2 9.85 8.99 5.66 66 44 11 2 44.7
55.12 44.1 2.89 7.521 8.56 5.42 4.88 6.789 5.423
7.1234 9.753 8.741 5 6
```

RESULTS:

```
5.5200,+01
7.1330,+01 7.7000,+01
```

Example 5. Newton's Method of Successive Approximations



Given: An area A defined by a circular arc of radius r and its chord.

Required: Find the value of angle x subtended by the arc.

Solution: The relationship between A and x is:

$$A = \frac{r^2}{2}(x - \sin x)$$

Like many practical problems, this one has no analytic solution. However, methods have been developed to find approximate solutions to such problems. The method to be used here is called Newton's Method. If the solution x to

$$f(x) = 0$$

is to be found, then a sequence of values approximating the solution x is given by

$$x_{n+1} = x_n - f(x_n)/f'(x_n).$$

For this problem

$$f(x_n) = (1/2)r^2(x_n - \sin x_n) - A$$

and

$$f'(x_n) = (1/2)r^2(1 - \cos x_n).$$

Therefore, using elementary algebra, the approximation scheme is

$$x_{n+1} = x_n - \frac{x_n - \sin x_n - 2A/r^2}{1 - \cos x_n}$$

This equation is solved repeatedly, each time with the previous value of x_{n+1} substituted for x_n to compute a new value for x_{n+1}. The second term of the equation is the difference between successive approximations.

When this difference becomes less than some specified value, the sequence of approximations is said to have converged to a solution. The iteration procedure is then terminated and the problem is considered solved.

Practical considerations place a limitation on the number of iterations permitted. If the sequence of approximations does not converge within a prescribed number of iterations, the procedure is terminated and the approximate solution is rejected.

The conditions used in this example are:

Area = 1.5

Radius = 5.0

The first approximation is $x_1 = 1.0$. The iteration procedure is then performed for a maximum of nine iterations. If the successive approximations differ by less than 0.00001, then the sequence of approximations is considered convergent. The iteration procedure is then terminated and the sequence of approximations and differences is printed out in the form of a table. Otherwise, the program is terminated with no output.

The following identifiers in the program represent the corresponding physical quantities:

AREA	Area enclosed by chord and arc (A)
RADIUS	Radius of circle (r)
ANGLE	Approximation to the angle x
CHANGE	Difference between successive approximations
SMALL	Criterion for convergence
G	For convenience, the quantity $2A/r^2$

The program is as follows:

```
BEGIN
   COMMENT                        EXAMPLE 5
            SAMPLE PROGRAM USING UNIVAC 1108 ALGOL $
   REAL AREA, RADIUS, SMALL, G $
   INTEGER I, K $
   REAL ARRAY ANGLE(1:10), CHANGE(1:9) $
   FORMAT F10(X9,'ITERATION',X5,'ANGLE',X9,'CHANGE',A1.1),
          F11(X13,I1,D15.6,D14.5,A1),
          F12(X9,'THE ITERATION PROCEDURE HAS CONVERGED',A1) $
   COMMENT SET UP VALUES TO BE USED IN PROBLEM $
     AREA = 1.5 $
     RADIUS = 5.0 $
     SMALL = 1.0&-5 $
     G = (2.0*AREA)/(RADIUS**2) $
   COMMENT  BEGIN ITERATION LOOP -- MAXIMUM OF 9 ITERATIONS $
     ANGLE(1) = 1.0 $
     FOR I = 1 STEP 1 UNTIL 9 DO
       BEGIN
       COMMENT COMPUTE CHANGE IN APPROXIMATE SOLUTION $
         CHANGE(I) = (ANGLE(I)-SIN(ANGLE(I))-G)/(1.0-COS(ANGLE(I))) $
       COMMENT TEST FOR CONVERGENCE OF APPROXIMATE SOLUTION $
         IF ABS(CHANGE(I)) LSS SMALL THEN GO TO L110 $
       COMMENT APPROXIMATION HAS NOT CONVERGED - COMPUTE NEXT
               APPROXIMATION $
         ANGLE(I+1) = ANGLE(I) - CHANGE(I)
       END $
   COMMENT  END OF LOOP - ITERATION PROCEDURE HAS NOT CONVERGED $
     GO TO FIN $
   COMMENT  THE ITERATION PROCEDURE HAS CONVERGED $
L110: WRITE (PRINTER,F10) $
     WRITE (PRINTER,F11, FOR K=1 STEP 1 UNTIL I DO
           (K,ANGLE(K),CHANGE(K))) $
     WRITE (F12) $
   FIN:
END OF PROGRAM $
```

Note that a completely blank card gives a blank line in print.

The sample gave the following result:

```
ITERATION      ANGLE           CHANGE
    1        1.000000          .08381
    2         .916186          .00742
    3         .908770          .00006
    4         .908714          .00000
THE ITERATION PROCEDURE HAS CONVERGED
```

This is in excellent agreement with the theory.

APPENDIX C.

JENSENS DEVICE

The purpose of this section is to acquaint the reader with two interesting programming techniques, namely Jensen's Device and Indirect Recursivity. A thorough treatment of the recursive concept may be found in "The Use of Recursive Procedures in ALGOL 60", H. Rutishauser *The Anual Review in Automatic Programming*, Pergamon Press, London, 1963.

Jensen's Device comprises the use of two parameters in a procedure call, in which one is a function of the other. Neither may be a value parameter.

The following example is a method of evaluating an approximation to the definite integral of a function by means of Simpson's Rule over one interval. The algorithm may be written:

```
REAL PROCEDURE SIMPS (X,ARITH, A, B) $
VALUE A,B $ REAL X, ARITH, A,B $
BEGIN REAL FA, FM, FB $
   X=A $ FA=ARITH $ X=B $ FB=ARITH $
   X=B-A)/2 $ FM=ARITH $
   SIMPS=(B-A)*(FA+4*FM+FB)/6
END SIMPSON INTEGRATION $
```

In a call of SIMPS, ARITH may be any arithmetic expression. Jensen's Device refers to the case when ARITH is a function of X. For example, the call:

```
I=SIMPS(Z,EXP(Z*Z), 0.0, 1.0)
```

would cause ARITH to be replaced by EXP(Z*Z) in the running program. This call evaluates an approximation to the integral

$$\int_0^1 e^{z^2}\, dz$$

In evaluating an approximation to the double integral

$$\int_0^1 \int_0^1 e^{xy}\, dy\, dx$$

indirect recursivity may be used by making the parameter corresponding to ARITH a call to SIMPS itself, thus

```
I=SIMPS(X,SIMPS(Y,EXP(X*Y), 0.0, 1.0), 0.0, 1.0)
```

More material may be found in: E.W. Dijkstra, *A Primer of ALGOL 60 Programming, Bound Variables*, Academic Press, London, 1962, pp. 57-59.

APPENDIX D.

DIFFERENCES BETWEEN NU ALGOL AND UNIVAC 1107/1108 ALGOL

1. Improvements

Note: The points below are not necessarily listed in order of importance.

1.1. User Convenience

a) Automatic resolution of type conflicts between actual and formal parameters.

b) Format phrases allowed in I/O statements.

c) Dynamic definition of format phrase parameters.

d) Local declaration not necessary.

e) New format phrases for: absolute positioning to column, centerjustified string, leftjustified integer and zero suppression.

f) Editing to and from a string in core (not using external devices).

g) Compilation of several external procedures in same element.

1.2. Diagnostics.

a) Improved check of legality of format phrases.

b) Improved error detection and recovery giving more precise message, eliminating superfluous and misleading diagnostics.

c) Undefined labels are detected on first reference not at the end of the program.

d) Warnings are given for inefficient use of language and legal but possibly dangerous constructions.

e) Full control at compile time of non formal and non external procedure parameter call, both number of parameters and type-kind-correspondance.

f) Control of number of subscripts for arrays at compile time.

1.3. Run-time Efficiency

a) Full utilization of all accumulators if necessary.

b) Inline arithmetic for all types.

c) Faster subscript mechanism including control of subscript range.

d) Improved procedure call mechanism with parameter control at compiletime.

e) Improved handling of formal parameters, short-circuiting the general mechanism for simpel name parameters when the type is correct.

f) All constant arithmetic performed at compiletime.

g) Improved addressing of non-local variables.

h) Improved addressing of formal name arrays providing efficient handling of vectors, matrices etc. in subroutines.

i) Double buffering of tape I/0.

j) Pseudo-evaluation of boolean expressions minimizing number of necessary tests in boolean expressions, especially useful in conditional statements.

k) Faster mechanism for calling FORTRAN subroutines.

l) Efficient handling of external machinecode procedures (EXTERNAL LIBRARY procedures) with full compiletime parameter check, and conversion capabilities for the parameters.

2. Changes and restrictions

2.1. External procedures

a) External procedures compiled using the UNIVAC 1107/ 1108 ALGOL compiler cannot be run together with ALGOL programs compiled using the NU ALGOL compiler (and vice versa).

b) External procedures compiled using the NU ALGOL compiler must have an E-option on the compiler control card (ALG card).

c) The declaration EXTERNAL NON-RECURSIVE PROCEDURE is not allowed.

d) The declarations for external procedures coded in SLEUTH II are EXTERNAL SLEUTH PROCEDURE or EXTERNAL LIBRARY PROCEDURE depending on the type of parameter transmission.

e) When using external FORTRAN procedures which have DOUBLE PRECISION or COMPLEX arithmetic, F-option must be used on the XQT card to avoid the run time error: 'ILLEGAL OPERATION'.

2.2. Declarations

a) The declaration OWN is not allowed.

b) The declaration OTHERWISE is not allowed.

c) Reserved ALGOL words cannot be used as variable names. Two new reserved words have been introduced: OPTION and OFF.

d) A procedure may have at most 63 parameters.

2.3. Formats

a) In input or output statements, the format identifier must come before the list to which it applies.

b) The format phrase T is not allowed.

2.4. Standard Procedures

1. The following changes have been made in the names of some of the standard procedures.

OLD	NEW	MEANING
COMPLEX	COMPL	Produce a complex number using the first parameter as the real part, and the second as the imaginary part.
IMAGINARY	IM	Obtain the imaginary part of the complex number given as parameter.
INTEGER	INT	Convert to type INTEGER.
REAL	RE	Obtain the real part of the complex number given as parameter.

2. The argument of a standard procedure is regarded as being by value.

2.5. FOR Statements

1. The controlled variable may only be of type REAL or INTEGER.

2. If the controlled variable is a subscripted variable, the subscript will keep the value that it had at the beginning of the FOR statement even if the statements controlled by the FOR change this value.

   Example:

   ```
   I = 3$
   FOR A(I) = (1,1,100) DO I = I + 1$
   ```

   When the FOR statement is finished

   ```
   A(3) will have the value 101
   I    will have the value 103
   ```

2.6 IF Statements

a) An IF statement after THEN must be enclosed with BEGIN END

b) An IF expression used in an arithmetic expression must be enclosed in parentheses.

Note: This is to eliminate the ambiguity of the "dangling else" and is clearly stated in the ALGOL 60 report.

2.7 Miscellaneous

a) All programs with the exception of external procedures must be enclosed with BEGIN END$

b) In a multiple assignment statement all of the variables to which the assignment is being made must be of same type.

c) The value specification must be placed in front of the type specifications.

d) Use of the device DRUM is somewhat different. See sec. 8.3.7.

e) In input and output, tapes 21 and 27 are no longer implemented. Continuous reading and re-reading may be done as shown in sec. 8.3.4.

f) The statement REWINT(TAPE())$ must be used instead of REWIND(TAPE(),INTERLOCK)$

g) When errors or EOF-conditions are detected during I/0 and no labels are provided, the program is terminated with an appropriate message.

h) Positioning to a KEY is halted if an EOF is encountered. Sec. 8.5.7.

APPENDIX E.

SYNTAX CHART.

Table of Contents.

INTRODUCTION

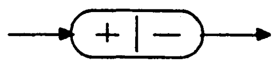
This appendix summarizes the syntax of NU ALGOL in chart form.

The use of the chart portion of the manual is very simple and almost self-explanatory. At the top of each page is a square box which contains the name of the concept defined on that page, for example,

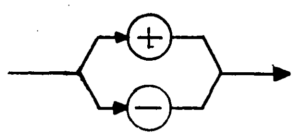
$\boxed{\text{type declaration}}$ ::=

The definition consists of a series of boxes connected by lines indicating the flow of symbols which define the concept. Two kinds of boxes are distinguished: those with round corners (or circles) and those with square corners. The round cornered boxes contain symbols that stand for themselves. Square cornered boxes contain names of concepts which are defined elsewhere in the chart and may be found by a quick reference to the index.

In some places a metalinguistic "or" symbol has been used (for reasons of space) and should be understood as follows:



is equivalent to



In some sections a pair of letters may mark two spots in a definition. Underneath that section will appear that letter pair followed by a name. This means that that name will be used in lieu of the string of symbols between the letter pair in other parts of the chart.
This chart uses only one of the two possible representations for some symbols in Algol. The following equivalences should be noted:

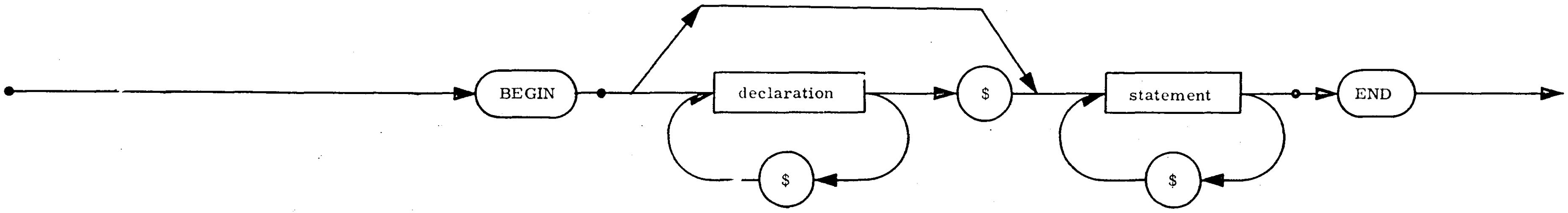
Symbol used in this chart	Alternate representation
(	[
)	]
:	..
=	:=
GO TO	GO or GOTO
$	;

In addition, comments may be inserted in the program by means of the following equivalences:

$ COMMENT ⟨any sequence not containing a $⟩ $ equivalent to $
BEGIN COMMENT ⟨any sequence not containing a $⟩ $ " " BEGIN
END ⟨any sequence not containing END or ELSE or $⟩ " " END

This chart makes no mention of the use of spaces within Algol. A space has no meaning in the language (outside of strings) <u>except</u> that it must not appear within numbers, identifiers, or basic symbols, and must be used to separate contigous symbols composed of letters or digits. Spaces may be used freely to facilitate reading.

program ::=

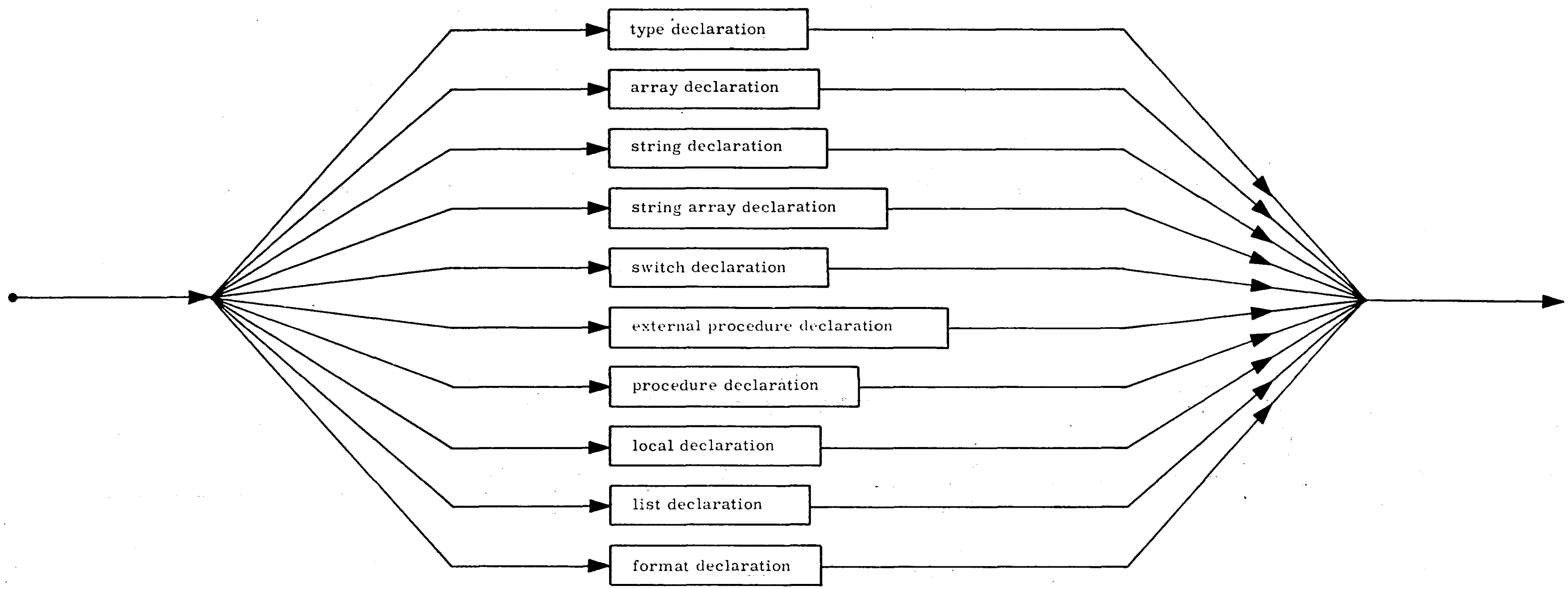


Explanation: A program is a complete set of declarations and statements which define an algorithm for solving a problem. The logic of this algorithm (its correctness) is the business of the programmer. The compiler only checks that the syntax (form) is correct.

Notice that the $ is used to separate declarations and statements and is not inherently a part of a declaration or statement. Nevertheless, it will be shown in most examples for clarity.

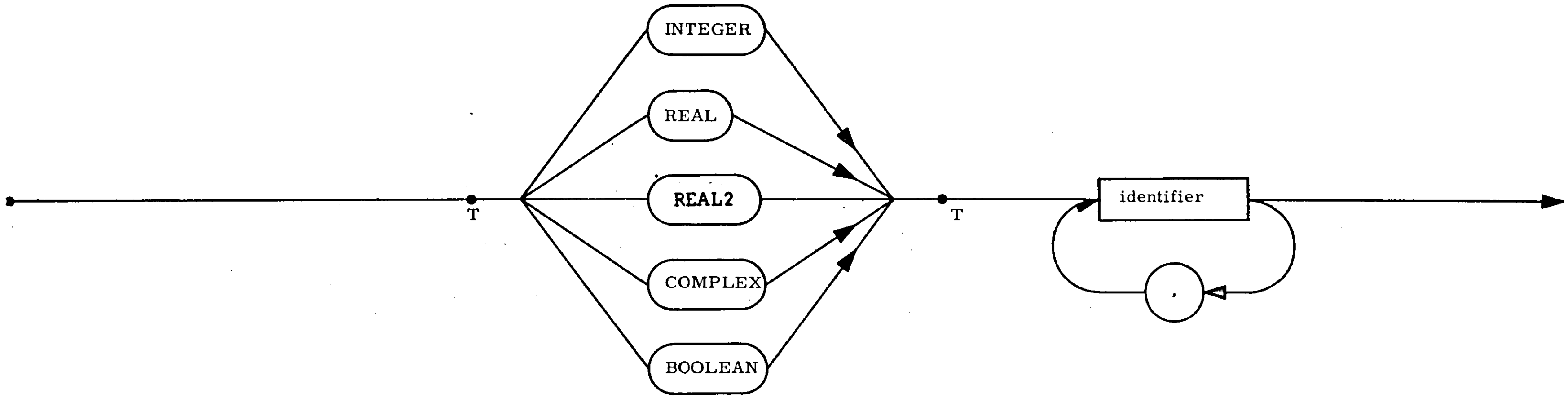
In an externally compiled procedure (E-option on the ALG card), the outermost BEGIN-END pair is not required.

declaration ::=



Explanation: There are 10 types of declarations each of which is defined in detail on the following pages.

type declaration ::=



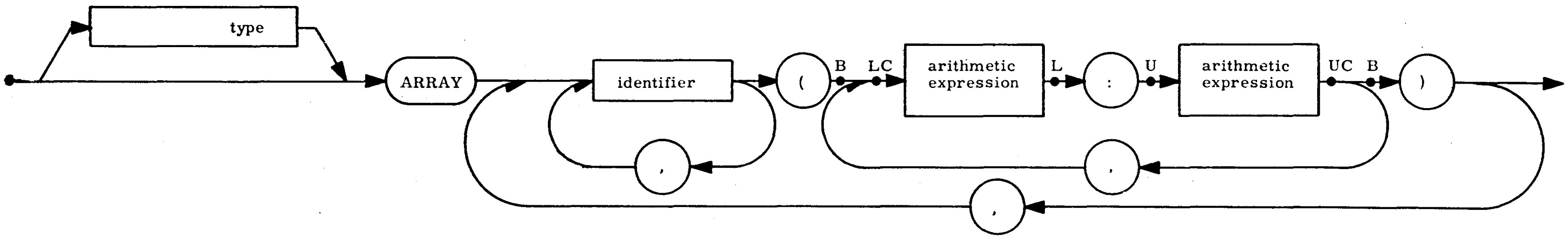
TT type

Explanation: A type declaration declares the mode of arithmetic the following identifiers will assume in the block. Types **REAL2** and COMPLEX associate 2 words with the identifier, the others one. Upon entrance to a block, identifiers are given the value zero.

Examples:

```
INTEGER I4, PAK, LOOPCNT $
      BOOLEAN ANYLEFT, LASTOUT $
COMPLEX C, CINVS $
REAL2 DP $
      REAL  QIN, QOUT, MAXITEM $
```

array declaration ::=

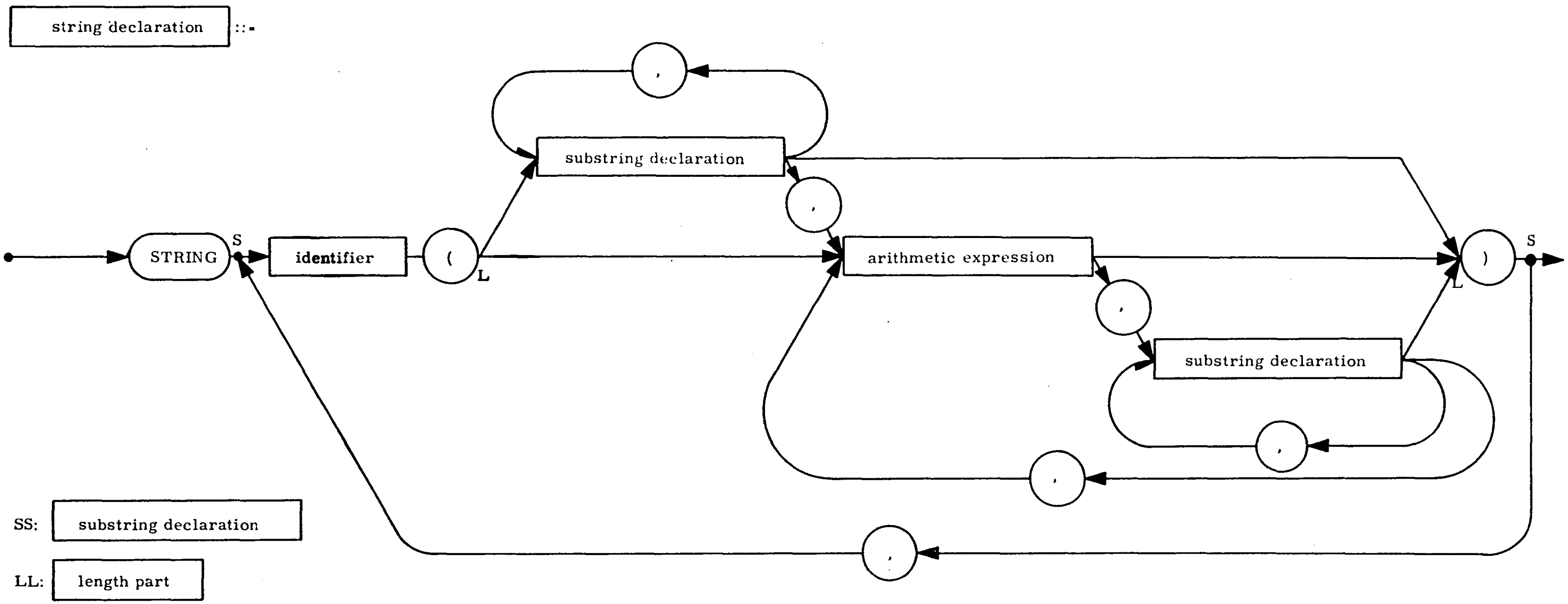


BB bound pair list   CC bound pair   UU upper bound   LL lower bound

<u>Explanation</u>: An array declaration associates an identifier with a 1-dimensional or larger matrix of values. The arithmetic expressions define the lower and upper limits of each dimension. The type plays the same role as for simple variables. If omitted, type REAL is assumed.

<u>Examples</u>:

```
COMPLEX ARRAY  CCON4 (0:N), CP1(1:N+1) $
BOOLEAN ARRAY BAND, BOR, BXOR(-4:4) $
REAL ARRAY     B(I-1:I+1), XINITIAL, YINITIAL(-N:N, -N:N, 1:2) $
      INTEGER ARRAY I(1:5), J, K, L(ENTIER(X): P112) $
ARRAY XYZ4(1:N×2) $
```

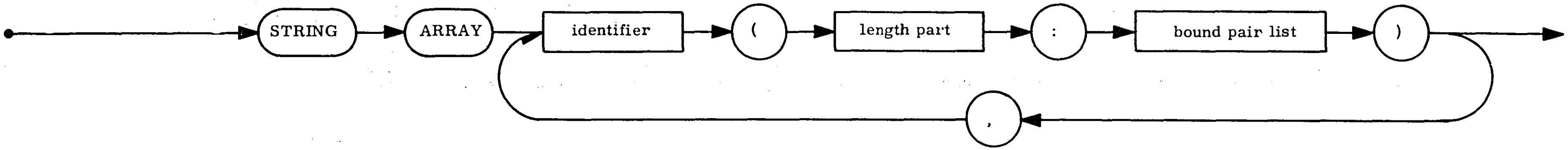


Explanation: A string declaration associates an identifier with a variable whose value is a string of characters. The length of the string is its number of characters. A group of characters of a string may be named as a substring. The length of a string must be less than 4096.

Examples:

```
STRING ST1(36),NAME(INITIALS(2),LAST(16)) $
STRING PI(N+2),QUOTE(1) $
      STRING NEXTOUT(80) $
STRING ALPHA(BETA(2,GAMMA(4),2),DELTA(EPSILON(6)),20) $
```

string array declaration ::=

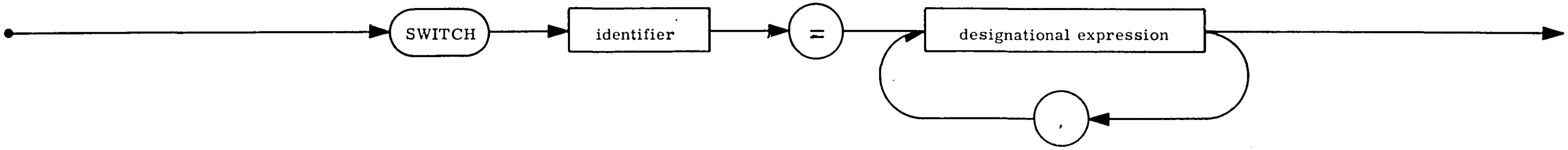


Explanation: A string array is a matrix whose elements are strings. Appended to the length part of the declaration are the bound pairs for each dimension, just as for an ordinary array.

Examples:

```
STRING ARRAY SA(80:0:100), CARD(LABEL(8), OP(6), 2, OPERAND(64):1:N) $
        STRING ARRAY LASTFILE (CLENGTH:1:507) $
```

switch declaration ::=

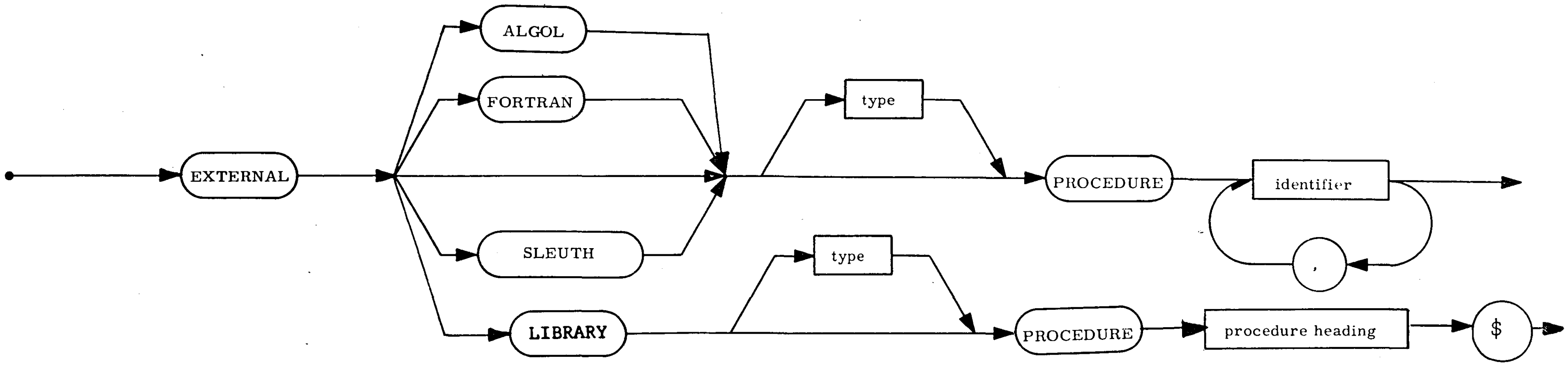


Explanation: A switch declaration associates an identifier with an ordered list of designational expressions. A switch is used for transfer to a label depending on the value of some variable.

Examples:

```
SWITCH JUMP = L1,START,FEIL4,SLUTT $
SWITCH BRANCH = IF BETA EQL 0 THEN L1 ELSE JUMP(J),START $
```

external procedure declaration ::=

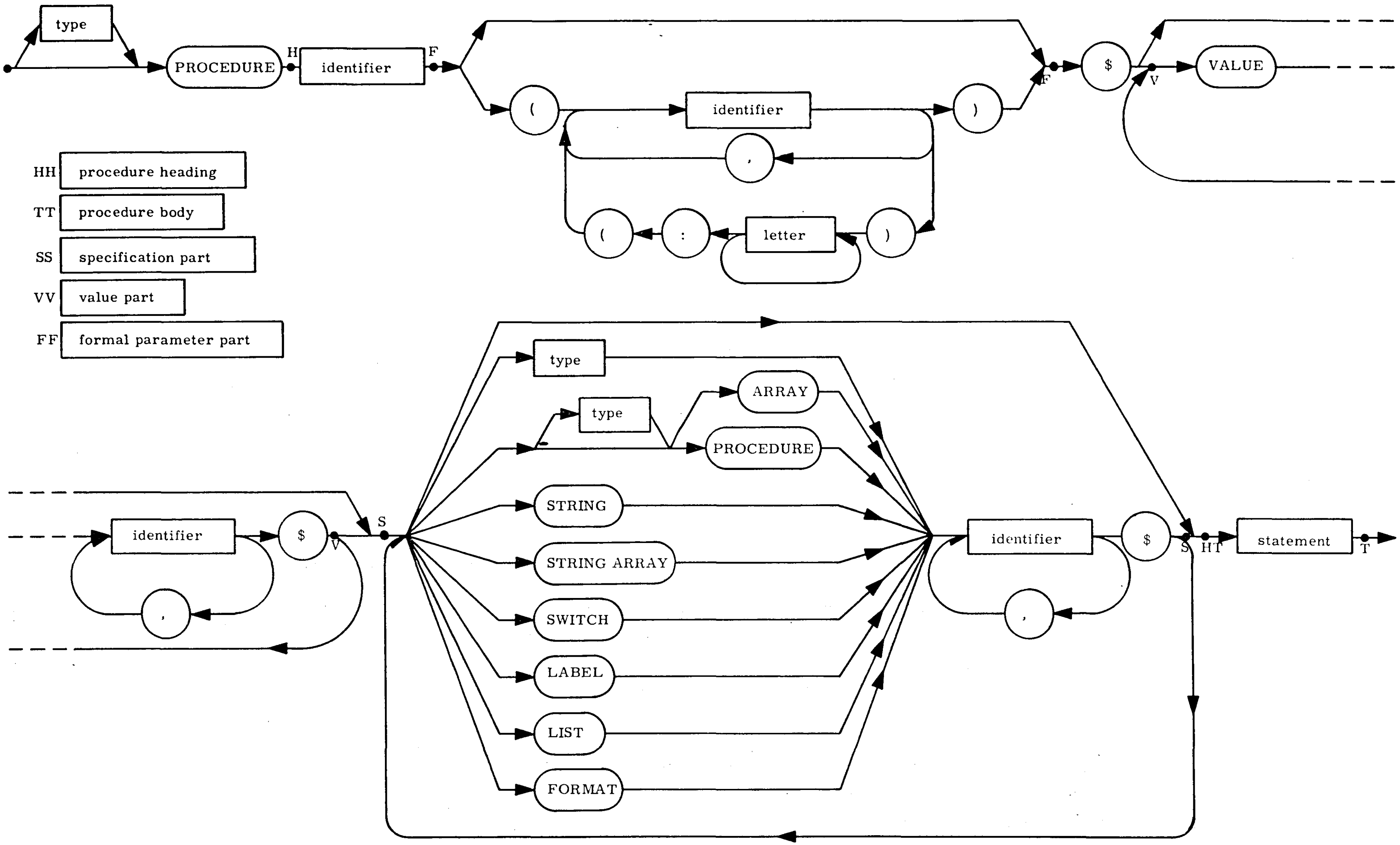


Explanation: This declaration specifies a list of identifiers which are to be the names of procedures not found in the program. These procedures may be written in assembly language (SLEUTH, LIBRARY), ALGOL or FORTRAN. The type of external procedures is specified if they are functional procedures.

Examples:

```
EXTERNAL FORTRAN REAL PROCEDURE CBRT$
EXTERNAL FORTRAN PROCEDURE NTRAN,INVS$
EXTERNAL PROCEDURE ROOTFINDER,KEYIN,KEYOUT$
EXTERNAL SLEUTH PROCEDURE TYPEIN,TYPEOUT$
EXTERNAL LIBRARY INTEGER PROCEDURE PACK(A,B,C)$
                         VALUE A,B$
                         INTEGER A,B,C$ $
```

procedure declaration ::=

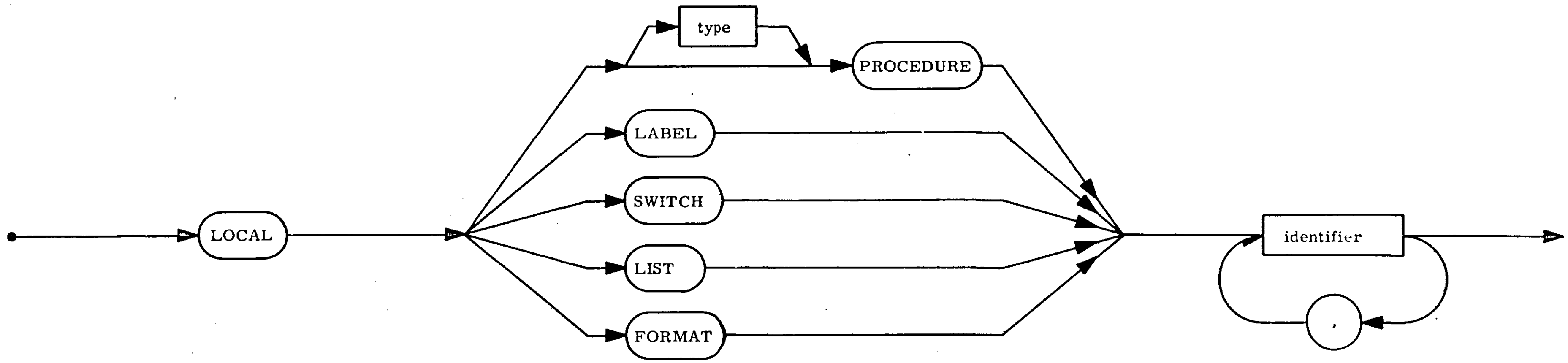


Explanation: A procedure declaration defines an algorithm to be associated with a procedure identifier. The principal constituent of a procedure declaration is a statement which is executed when the procedure is "called" (see procedure statement and function designator). The procedure heading specifies that certain identifiers appearing whithin the procedure body are formal parameters. A parameter may also be specified as "VALUE" in which case the procedure statement, when called, has access only to the value of the corresponding actual parameter, and not to the actual parameter itself.

Examples:

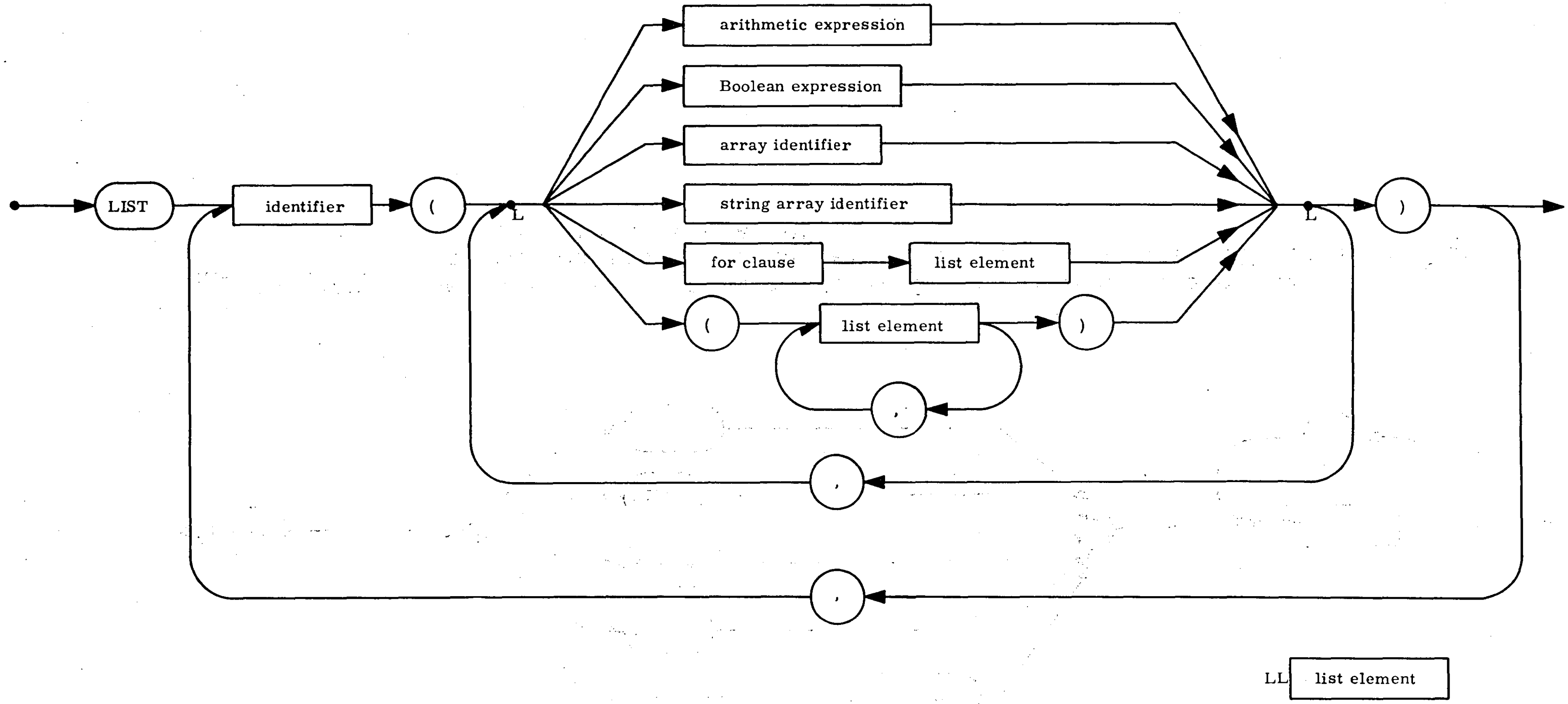
```
PROCEDURE ZEROSET (A,N) $
VALUE N $ INTEGER N $ ARRAY A $
BEGIN COMMENT THIS PROCEDURE ZEROES AN ARRAY ASSUMED DECLARED ARRAY A(1:N) $
INTEGER I $
FOR I = 1 STEP 1 UNTIL N DO A(I) = 0 END ZEROSET $

INTEGER PROCEDURE FACTORIAL (NUMBER) $
VALUE NUMBER $ INTEGER NUMBER $
 FACTORIAL = IF NUMBER LSS 2 THEN 1 ELSE NUMBER x FACTORIAL (NUMBER-1) $

BOOLEAN PROCEDURE BOOL $
BOOL = NOT (FINISHED AND OFF OR FIRST AND LAST) $
```

local declaration ::=



Explanation: The local declaration in NU ALGOL is treated as a dummy declaration and has been retained only for compatibility with the with the old UNIVAC ALGOL.

list declaration ::=

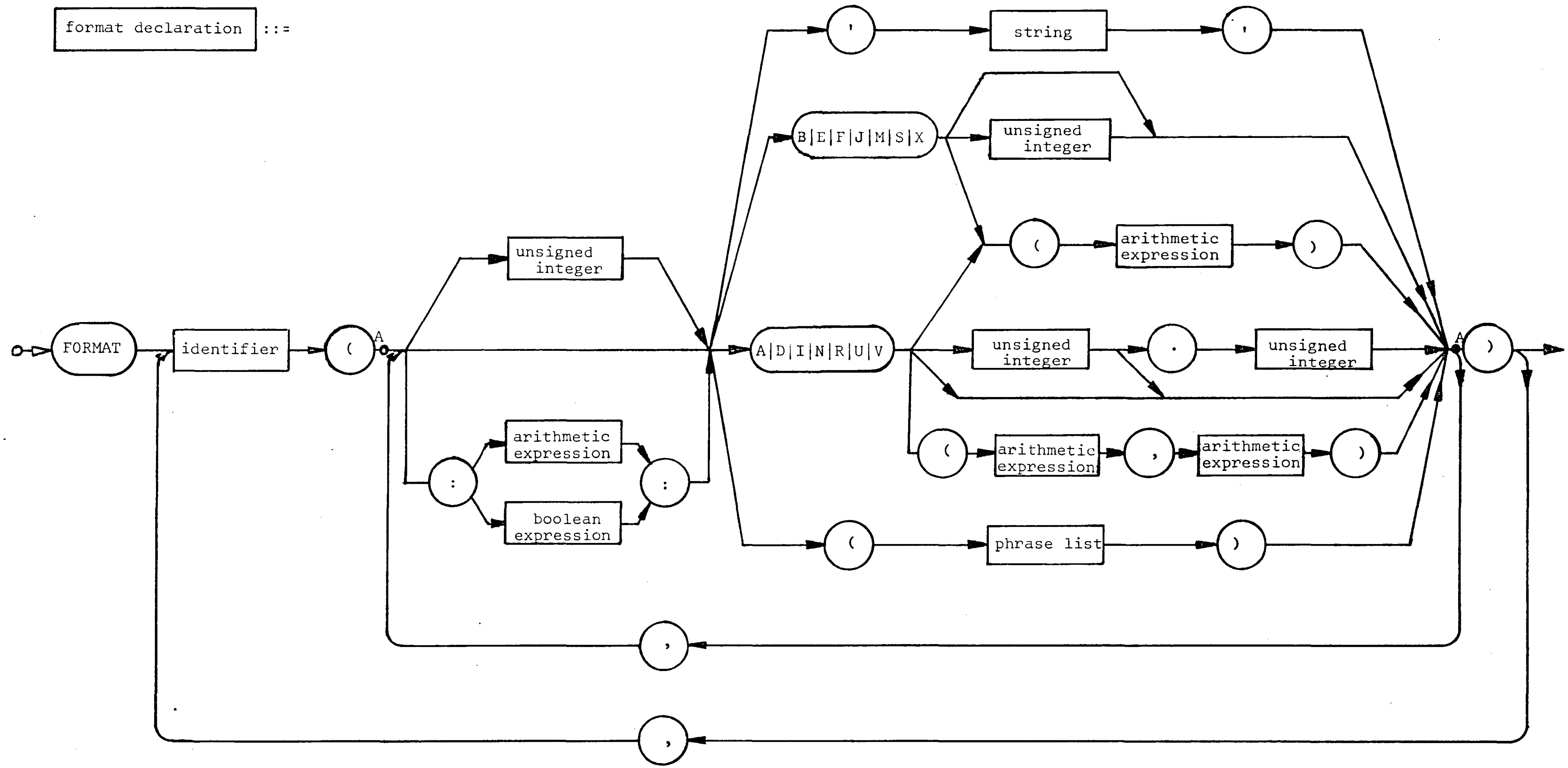


Explanation: A list defines an ordered sequence of expressions and array identifiers. A list may only be used as a parameter to a procedure, and, ultimately, only by a procedure written in non-Algol language.

Examples:

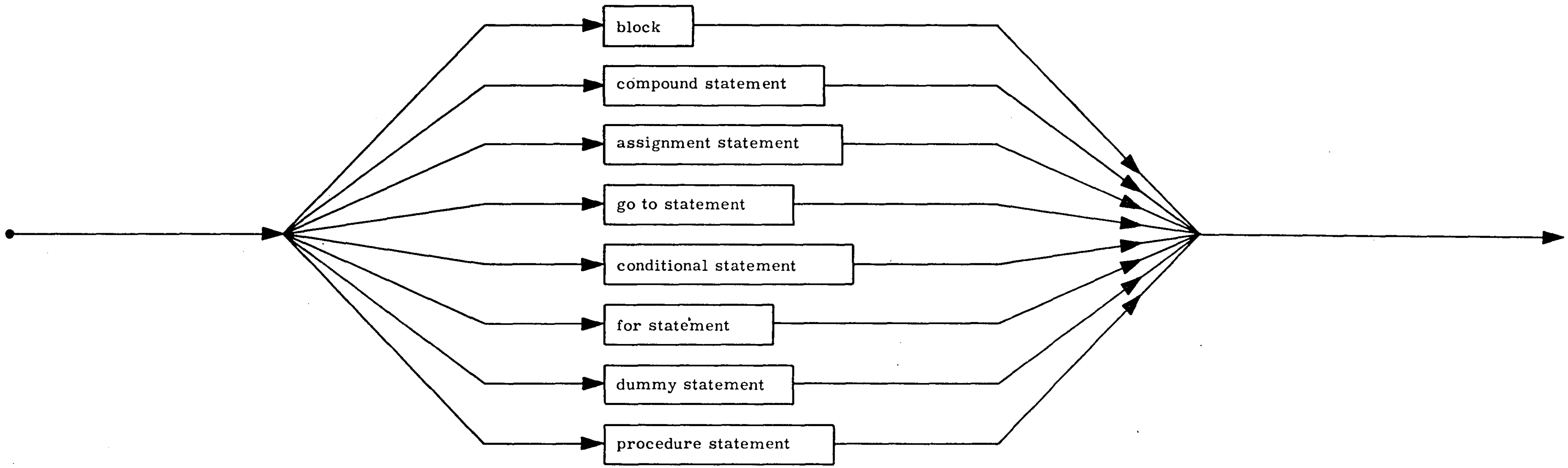
```
LIST OUT  (A+1, N+1, FOR I = (1, 1, NMAX)DO(Q(I), QRES(I))) $
LIST L1(A, B, C), L2(IF MOD(Q, 2)EQL 0 THEN B ELSE Q) $
```

format declaration ::=



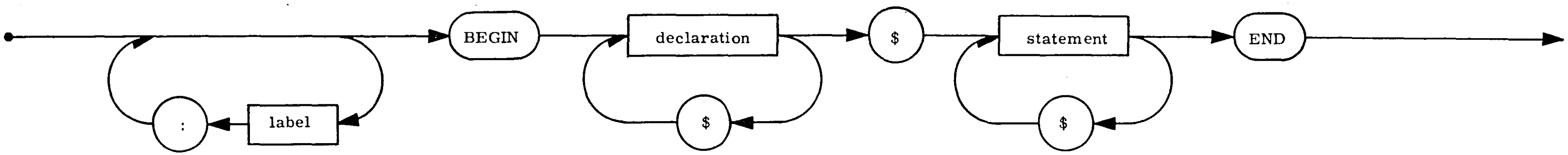
AA phrase list

statement ::=



Explanation: Statements define the sequence of operations to be performed by the program. The 8 types of statements are each defined in the following pages.

block ::=

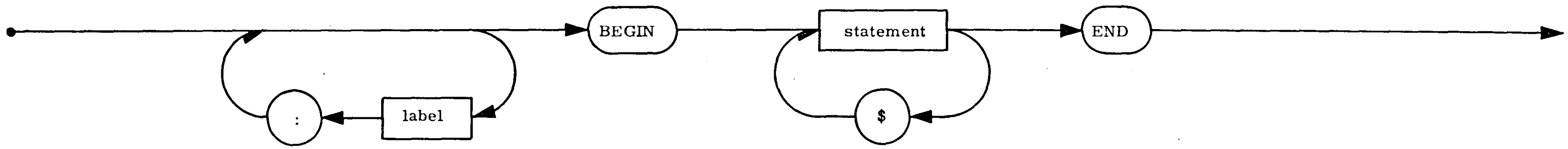


Explanation: A block automatically introduces a new level of nomenclature by a set of declarations. This means that any identifier declared in the block will have the meaning assigned by the declaration, and any entity represented by such an identifier outside the block is completely unaccessible inside the block. The identifiers declared within a block are said to be local (to that block) while all other identifiers are non-local or global (to that block).

Example:

```
L:BEGIN  INTEGER ARRAY A(1:10) $
         A(1) = 1 $
         FOR J = (2,1,10) DO A(J) = A(J-1) + J $
         FOR J = (1,1,10) DO WRITE (J,A(J)) $
END $
```

compound statement ::=

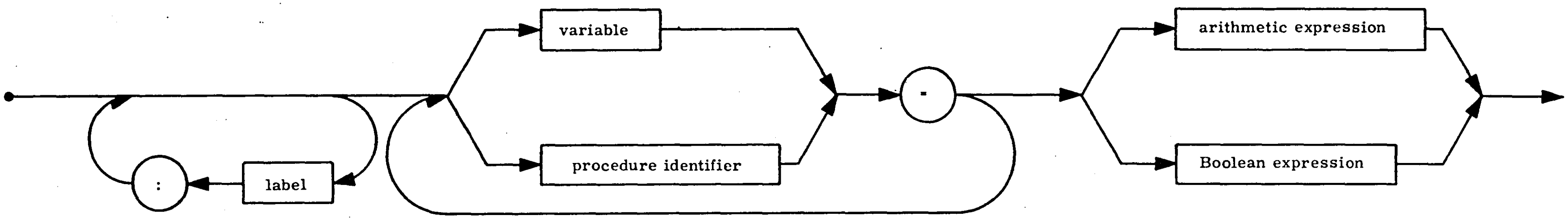


Explanation: A compound statement serves to group a set of statements by enclosing them with a BEGIN-END pair. This is then treated as a single statement.

Example:

```
BEGIN  T= 0 $ FOR I = 1 STEP 1 UNTIL M DO
          T= B(J,I) x C(I,K) + T $
          IF T GTR 820 OR OVFLOW THEN GO TO SPILL $
END $
```

assignment statement ::=

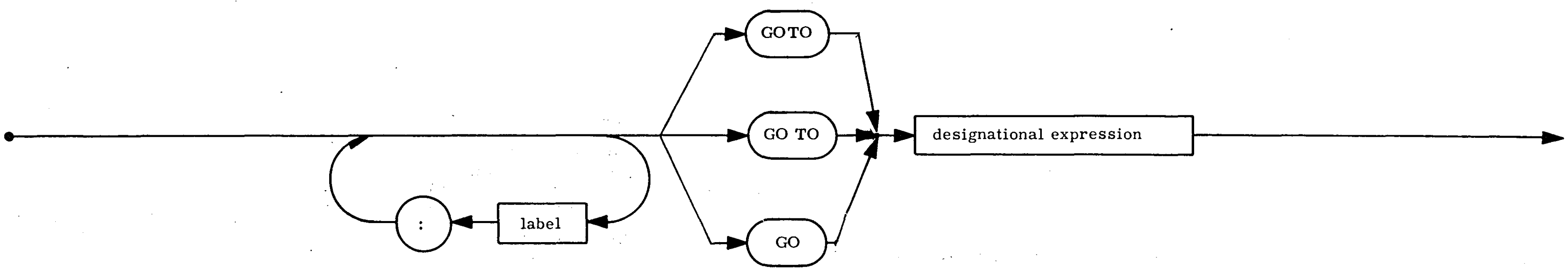


Explanation: An assignment statement serves to assign the value of the expression on the right-hand side to the variable and procedure identifiers on the left hand side. A procedure identifier is only permitted on the left-hand side in case the statement appears in the body of that functional procedure. If any of the left part variables are subscripted variables, they are evaluated before the expression is evaluated. Transfers of type are automatically evoked when necessary.

Examples:

```
A(I) = B(I) = &35 $
AANDB = A AND B OR EPS1 GEQ EPS2 $
P = SQRT(Bxx2 - 4xAxC) $
T = S - MYOxEPSOx(2xPIxF)xx2$
S(V,K-2) = COS(ANGLE) + 0.5 x(IF S1 THEN Kxx3 ELSE Kxx5) $
NAME(1,6:P + 1) = 'IFTHEN' $
```

go to statement ::=

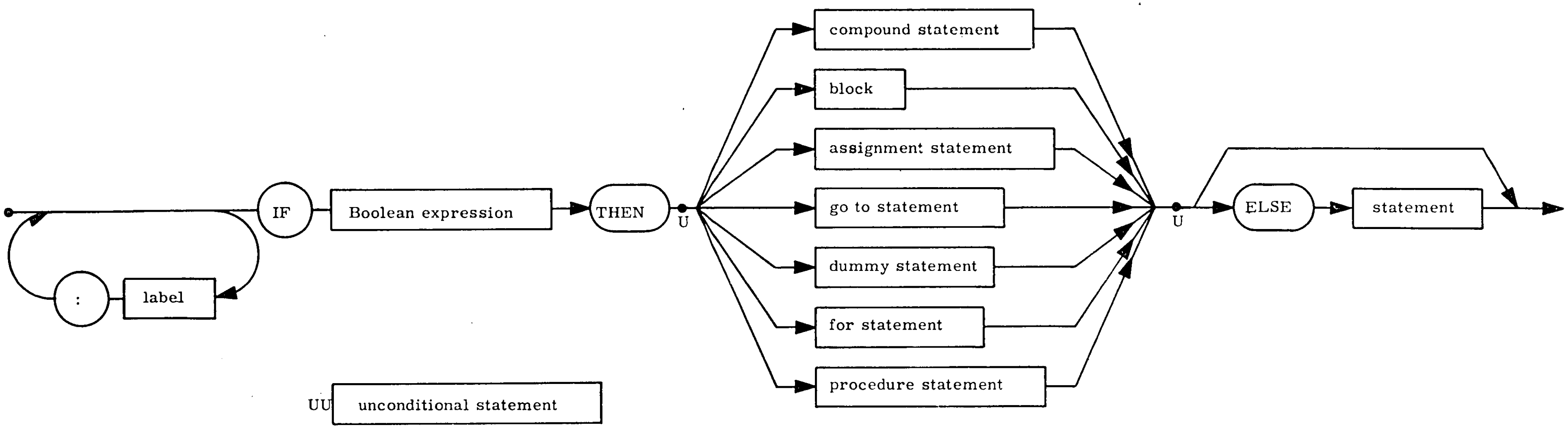


Explanation: A go to statement causes transfer of control to the statement with the label determined by the designational expression.

Examples:

```
GO TO PART4 $
GO TO OPS (I-2) $
GO TO IF ALPHA GTR 0 THEN Q17 ELSE JUMP(-ALPHA) $
GO TO TRACK (IF MOD(P, 2) EQL 1 THEN I ELSE A(I)) $
```

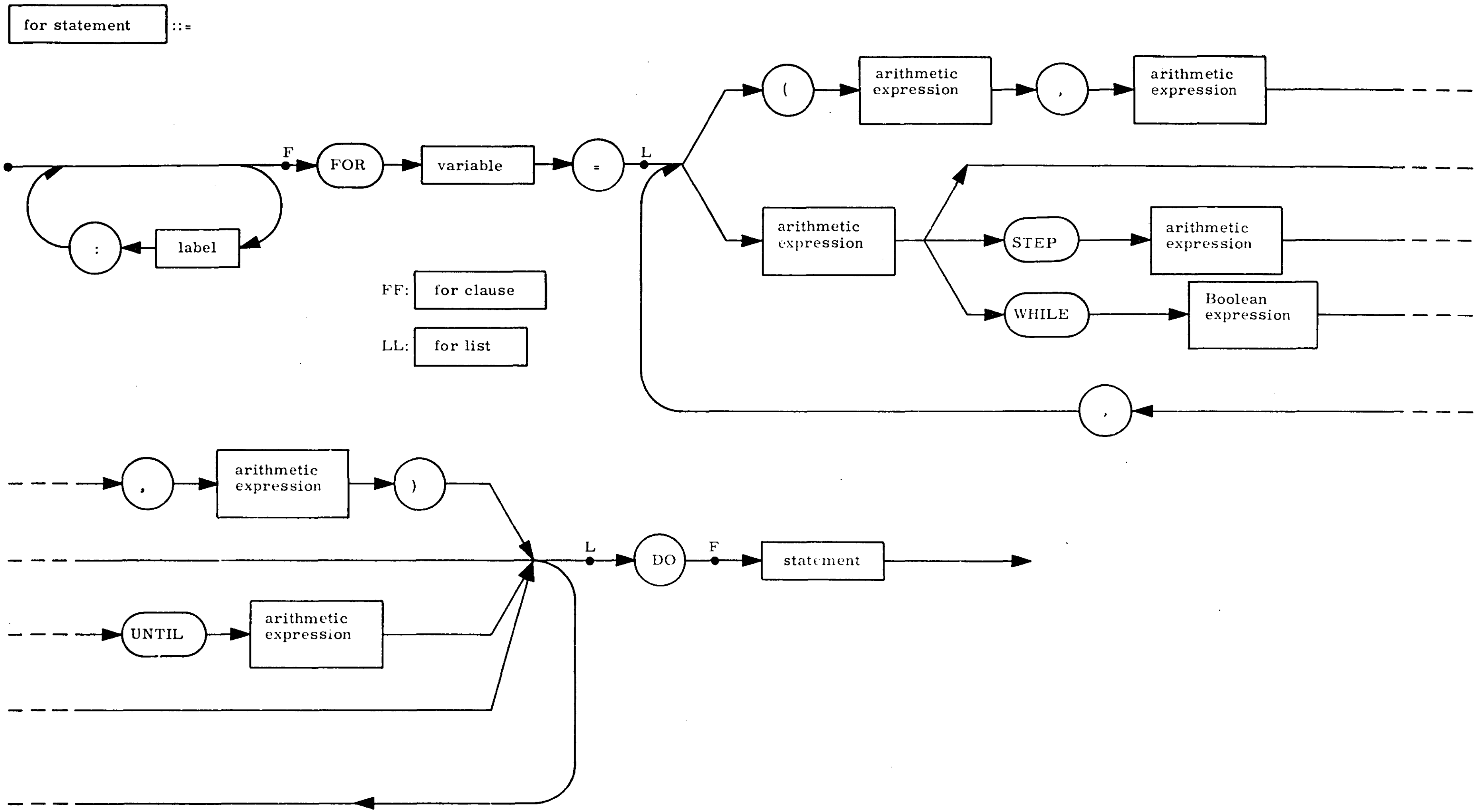
conditional statement ::=



Explanation: The if statement causes the execution of one of a pair of statements depending on the value of a Boolean expression. If this expression is TRUE then the statement after the THEN is executed and the statement after the ELSE is skipped. If FALSE, then the statement after the ELSE is executed, if it exists.

Examples:

```
IF C1 GTR 10 THEN A(0,0) = KMAX(I) ELSE GO TO LOOP $
IF BOOL(J) IMPL BOOL (J+1) THEN STEP(J) = 'VALID' ELSE STEP(J) = 'INVALID' $
IF I GEQ 0 THEN BEGIN FOR K = -1 STEP 1 UNTIL I DO B(K) = -COS(A-I) $
                     SUM = ADDUP(B) END ELSE
                     BEGIN IF I EQL -1 THEN GO TO ERROR ELSE GO TO NEXT END $
```

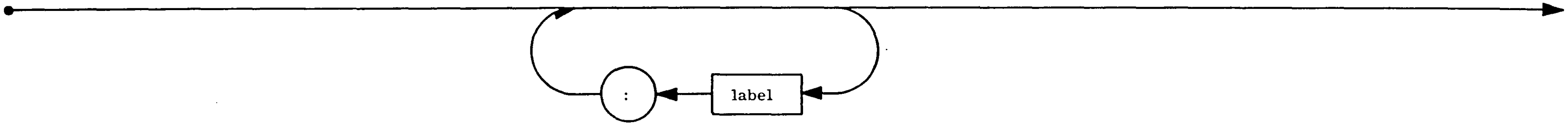


Explanation: The FOR statement controls the execution of the statement following the DO a number of times while the variable to the left of the = is assigned the values determined by the for list. The (,,) construction is equivalent to the STEP-UNTIL construction.

Examples:

```
FOR I = 1 STEP 1 UNTIL N DO FOR J = 1 STEP 1 UNTIL M DO A(I,J) = 0 $
FOR S - S + 1 WHILE P(S) NEQ 'A' AND S LEQ 80 DO BEGIN
                    N=Nx10 + P(S) $ IF OVFLOW THEN GO TO SIZERR END $
FOR S = (1,2xS-S,2xx10), -1,-2,-4 DO IF LOGAND(S,VAR) THEN GO TO YES $
```

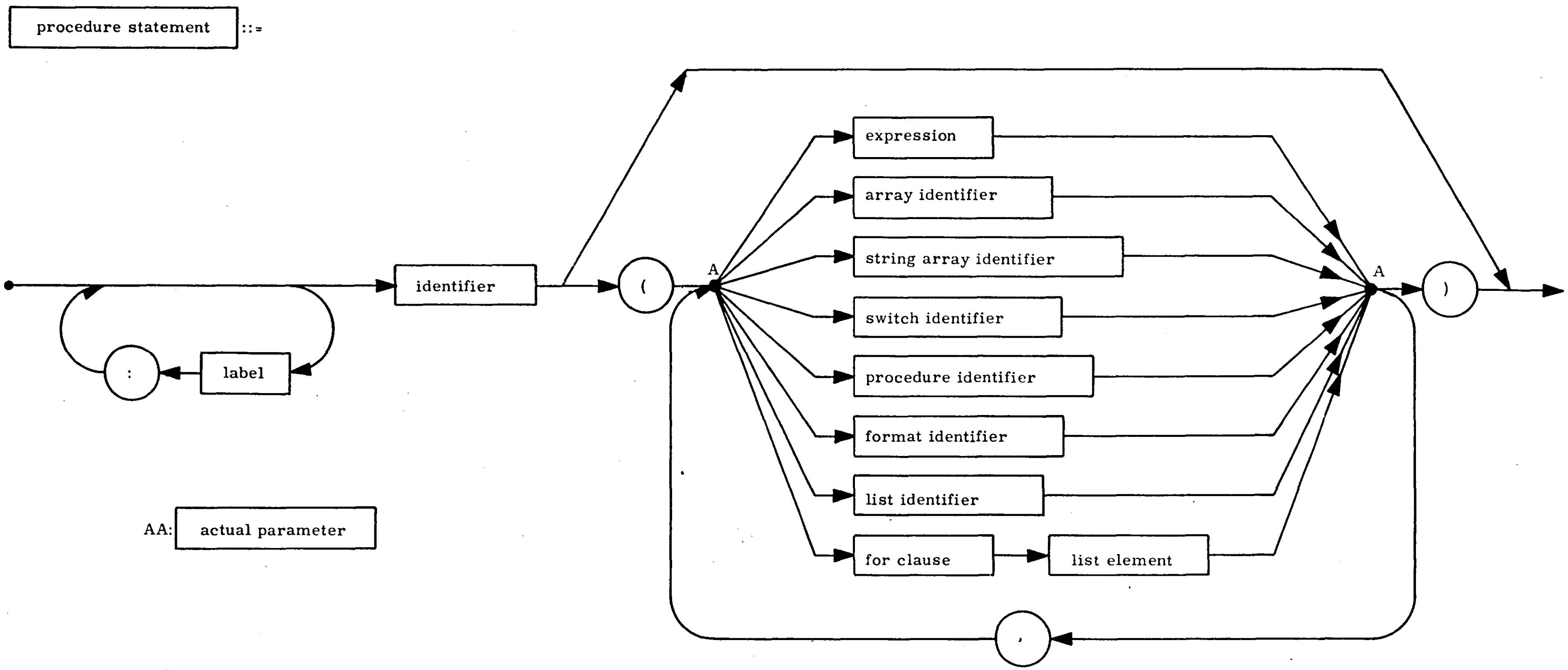
dummy statement ::=



Explanation: A dummy statement does nothing. It may serve to place a label.

Examples:

```
FOR I = (1, 1, N) DO FOR J = (1, 1, N) DO BEGIN
      IF I EQL J THEN GO TO ENDLOOP $
      :
      . . . $ ENDLOOP: END $

S = 0 $
FOR S = S + 1 WHILE P(S) NEQ 'A' DO $
```

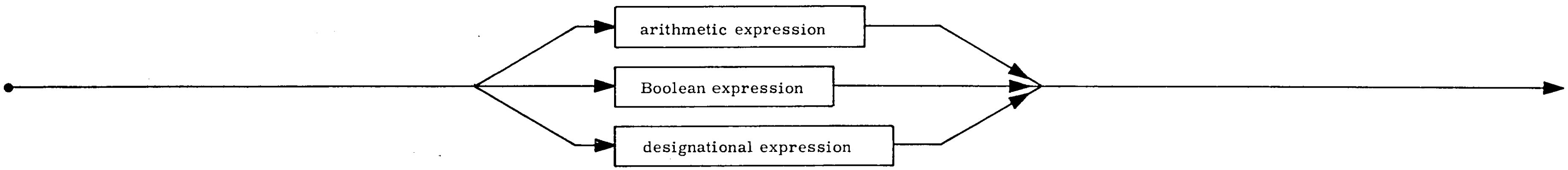


Explanation: A procedure statement is a call on a declared procedure. The actual parameters of the call replace the formal or dummy parameters throughout the body of the declared procedure. If the corresponding formal parameter has been "VALUE" specified then only the value of the actual parameter is used by the procedure.

Examples:

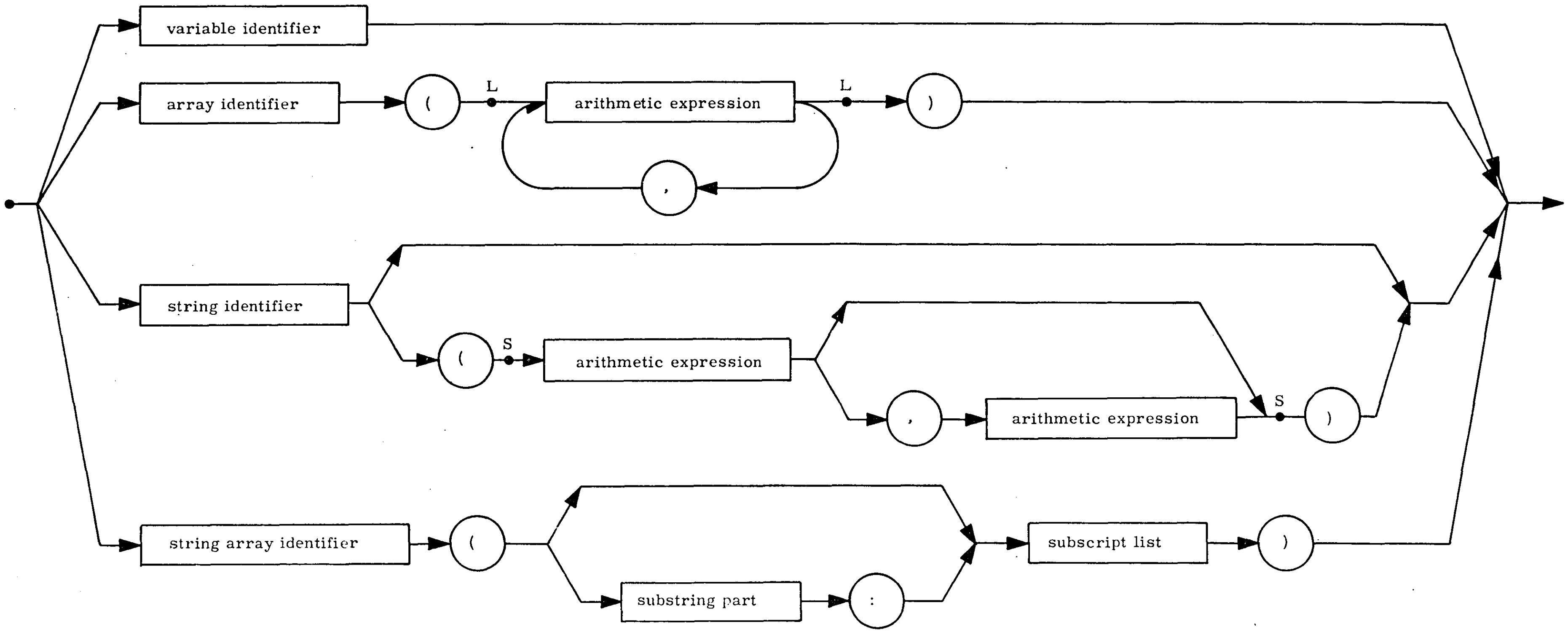
MARGIN (62, 56, 4) $
P(A, B, C, I, J, K) $
ROOTFINDER (N, O, ERGDET, KOEF, -4&&0, &&-5, 5. 0&&-1, 1000) $

expression ::=



Explanation: There are 3 types of expressions, classified according to their values. An arithmetic expression has a numerical value or a string value, a Boolean expression either TRUE or FALSE, and a designational expression has a label as a value.

variable ::=



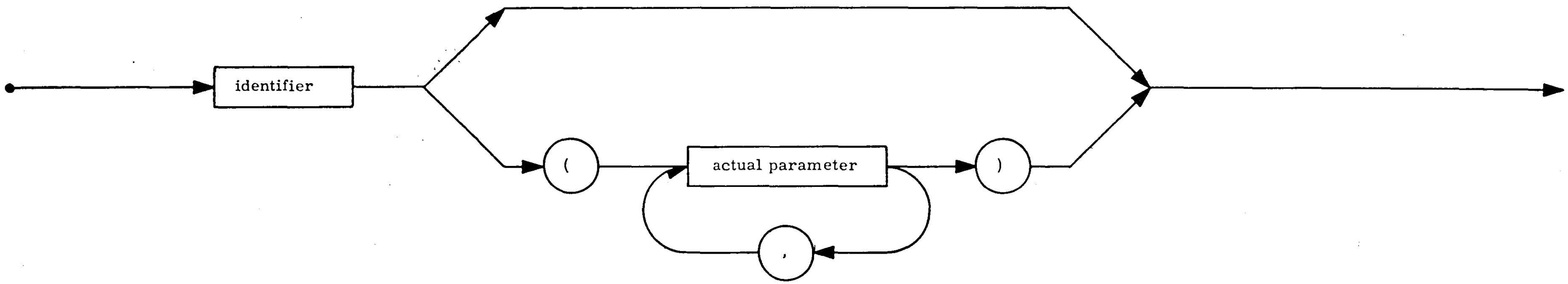
LL: subscript list

SS: substring part

Explanation: A variable is a designation given to a single value. A variable identifier is a variable named in a type declaration.

Examples:
DELTA
BOOLV(7)
CARD
CARD(4)
CARD(I, 6)
A(P(4), N*SIN(ANG), 3)
CUROUT(J, K)
CUROUT(1:J, K)
CUROUT(1, 6: J, K)

function designator ::=

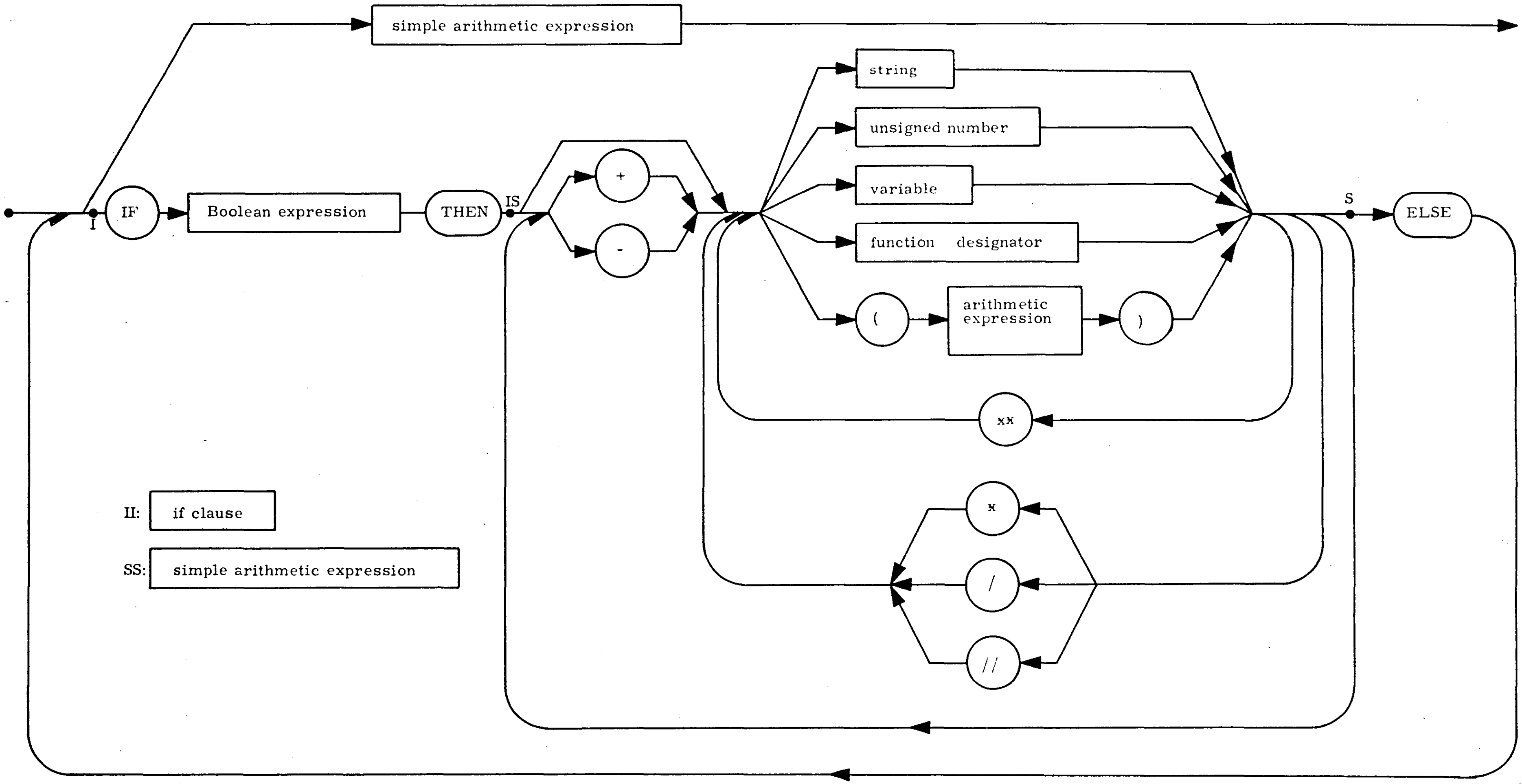


Explanation: A function designator defines a single numeric or logical value by applying the rules of the procedure declaration to the actual parameters. Only a procedure which has a type associated with it can be a function designator. Besides those functional procedures declared in the program, several standard ones are available for use without being declared.

Examples:

CLOCK
ARCTAN(1. 0)
BACKSLASH(A1, A2)

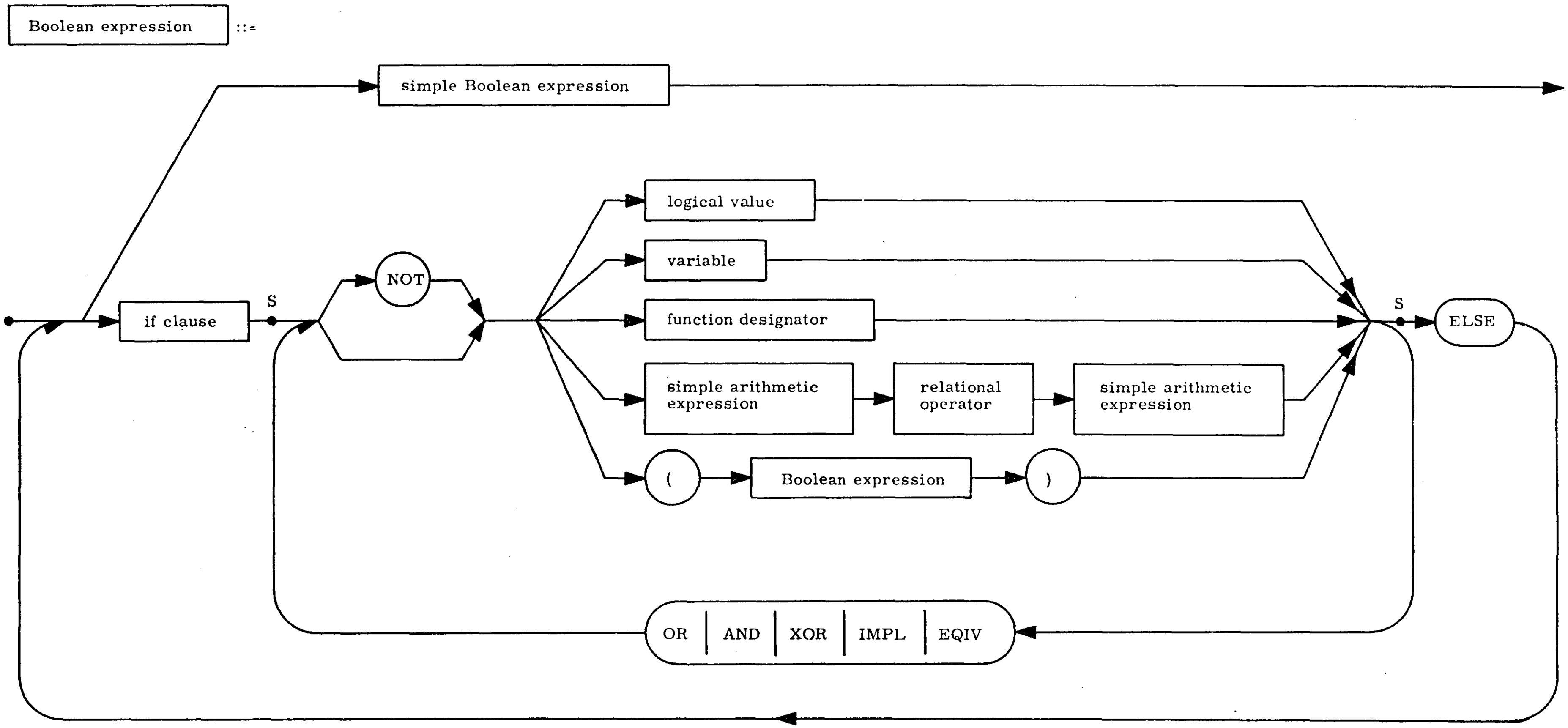
arithmetic expression ::=



Explanation: An arithmetic expression is a rule for computing a numerical value.

Examples:

```
A(4) + 2 x SQRT(Dxx3) - DELTA
IF A LSS &-5 THEN 0 ELSE A/&5
Q(MOD(N, 2) + 1) x (IF FIRST THEN 10 ELSE RATIO) // 3
```

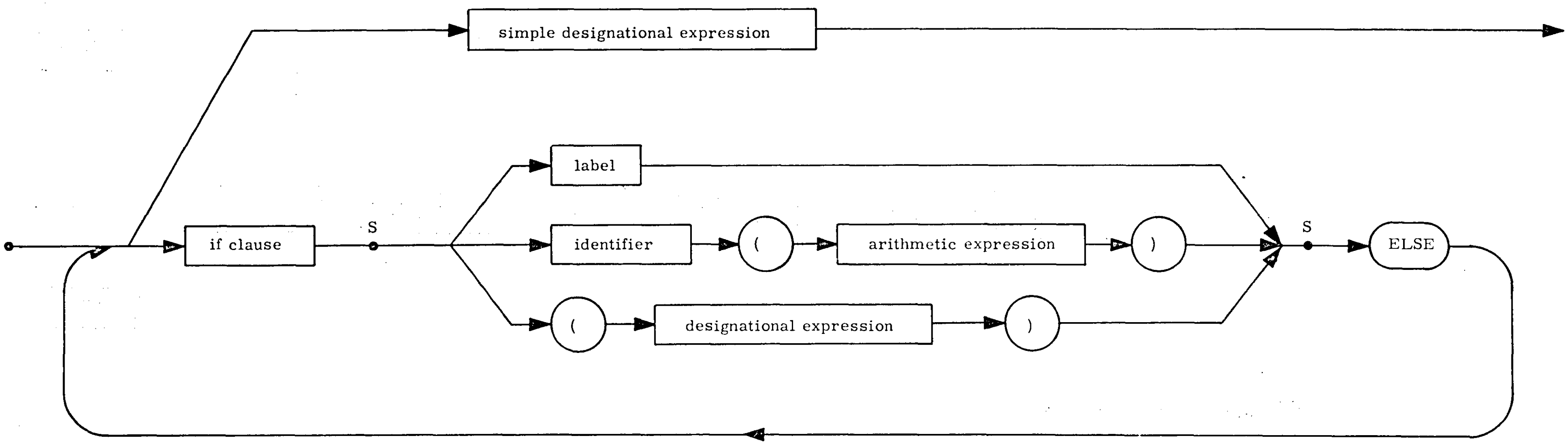


SS: simple Boolean expression

Explanation: A Boolean expression is a rule for computing a logical value.

Examples: FIRST AND NOT SPECIAL
A LSS DELTA OR ITERATIONS GTR MAXN
IF BETA THEN TRUE ELSE IF STEP(I) IMPL STEP(I+1) THEN QVALUE(P, I) ELSE QVALUE(P, I-1)

designational expression ::=

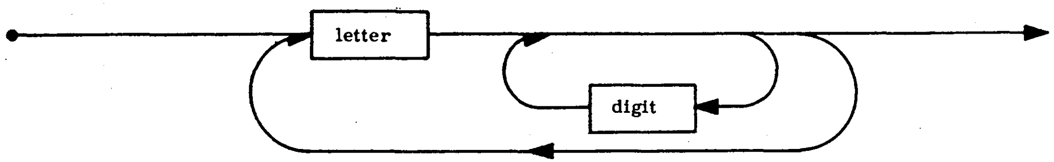


SS: simple designational expression

Explanation: A designational expression is a rule for computing the label of a statement. A switch identifier followed by an arithmetic expression in parenthesis refers to the label in the corresponding position in the switch declaration.

Examples: L10
IF BETA THEN SLUTT ELSE NEXT (K//2)

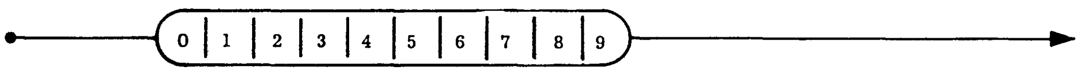
identifier ::=



variable identifier ::= array identifier ::=
string identifier ::= string array identifier ::=
switch identifier ::= procedure identifier ::=
list identifier ::= format identifier ::=
label ::= identifier

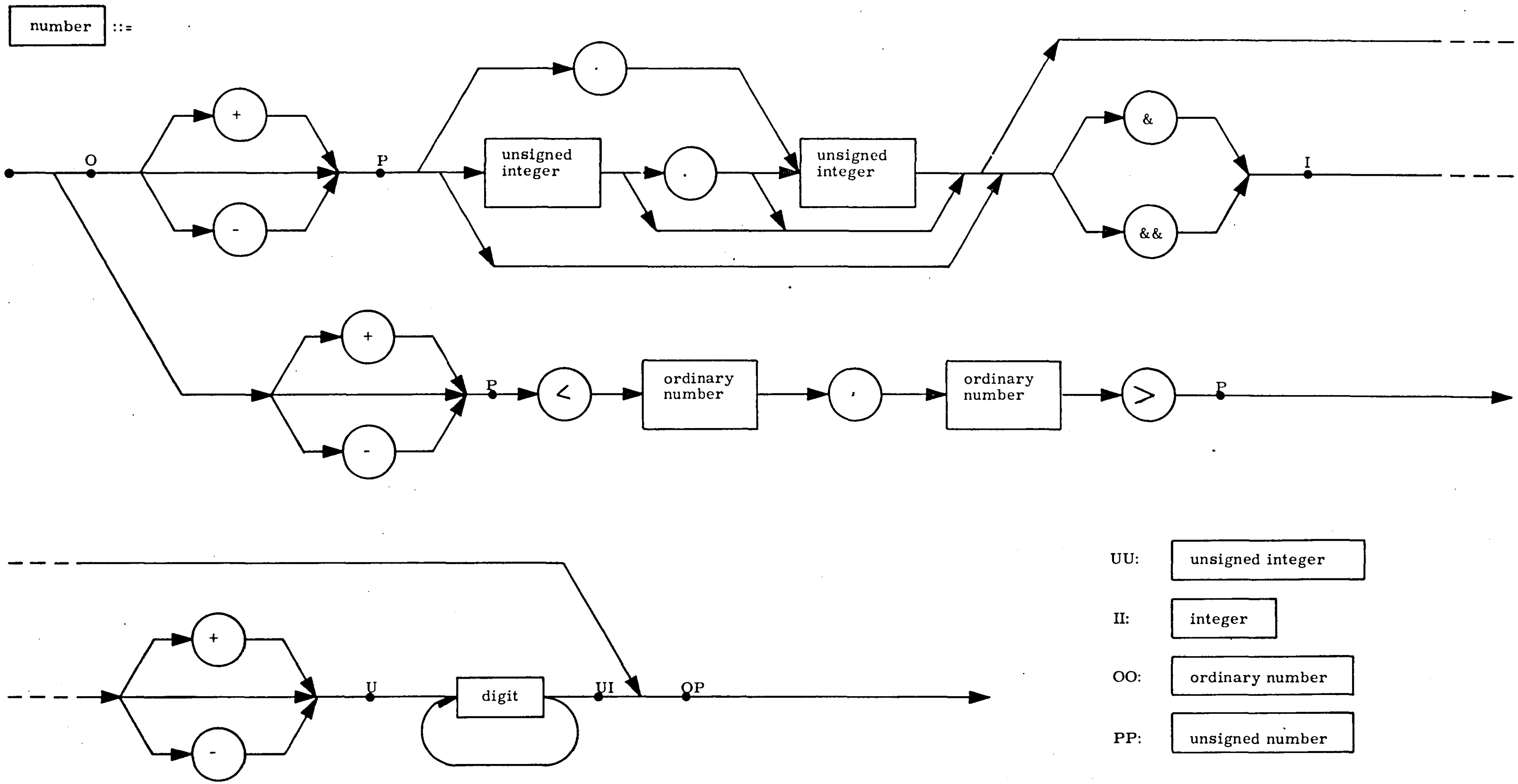
letter ::=

digit ::=



Explanation: An identifier is a name chosen to represent a variable, array, etc. Only the first 12 characters of an identifier uniquely define it.

Examples: P47
DELTA
SQRTROOOF2
E1C4PDQ

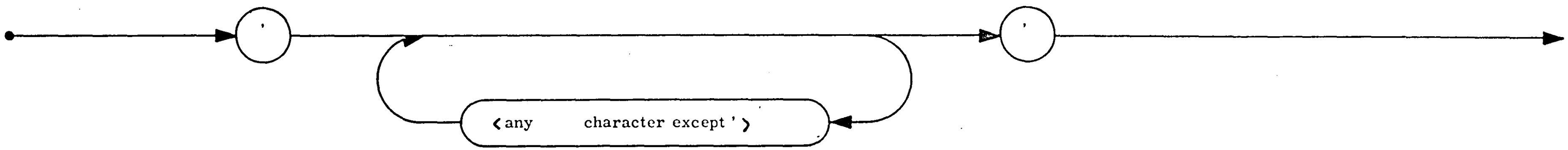

number ::=
+
-
.
unsigned integer
.
unsigned integer
&
&&
O
P
I
+
-
P
<
ordinary number
,
ordinary number
>
P
+
-
U
digit
UI
OP
UU: unsigned integer
II: integer
OO: ordinary number
PP: unsigned number

Explanation: A number is written in its usual decimal notation with the conventions of & for power of ten and corner brackets for complex numbers. Numbers are of 4 types: REAL, INTEGER, REAL2 and COMPLEX. REAL2 is differentiated from REAL by use of && for power of ten, or by having between 9 and 16 digits in the mantissa. COMPLEX numbers are distinguished by the corner brackets, where the first number is the real part and the second the imaginary.

Examples:

1
-1009
-.4031
3.1459
-18.0&4
-⟨1, 0⟩
20&-5
+1800.&&0
&-6
+⟨-.06, &-2⟩

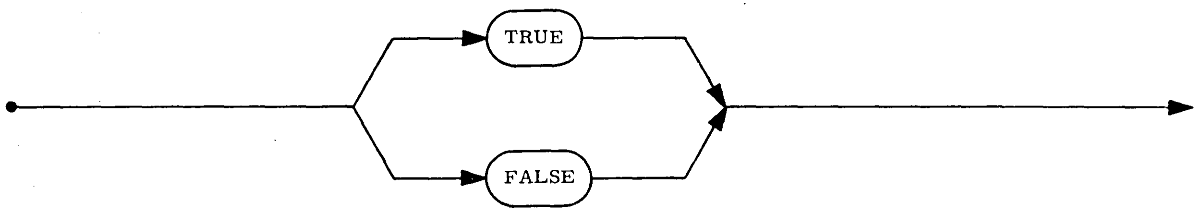
string ::=



<u>Explanation:</u> A string constant is any string of characters which are used as parameters to procedures or with string variables.

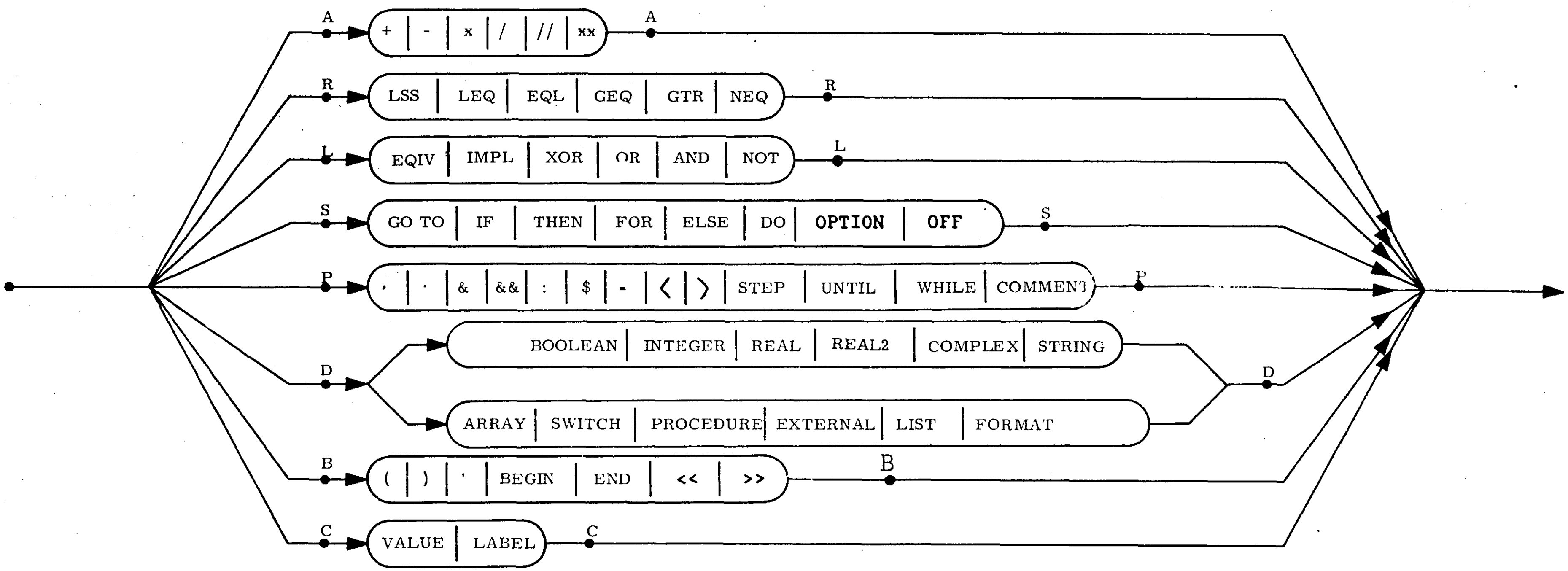
<u>Examples:</u>
'DOGGENBURG STR. 22'
'NEQ'
'BJARNE WIST'
'227 KALPHA'
' REAL ARRAY '

logical value ::=



<u>Explanation:</u> A logical value is a Boolean constant.

delimiter ::=



AA: arithmetic operator

RR: relational operator

LL: Boolean operator

SS: sequential operator

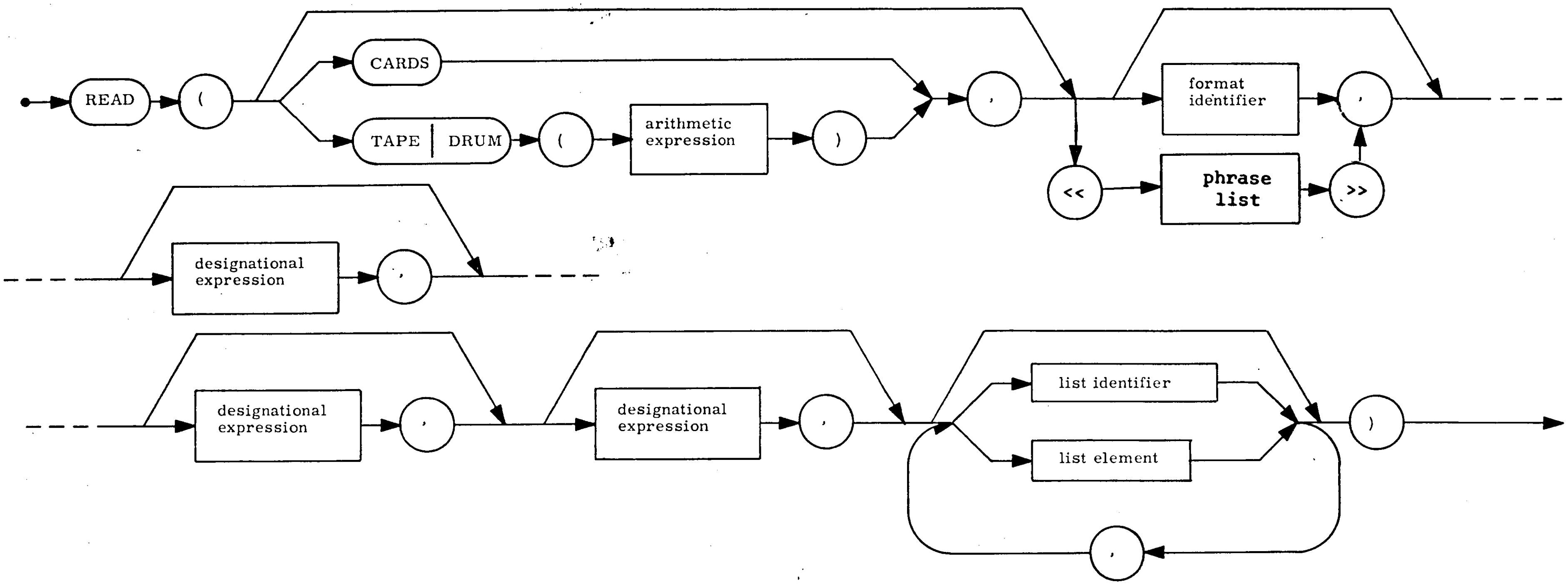
PP: separator

DD: declarator

BB: bracket

CC: specificator

input statement ::=

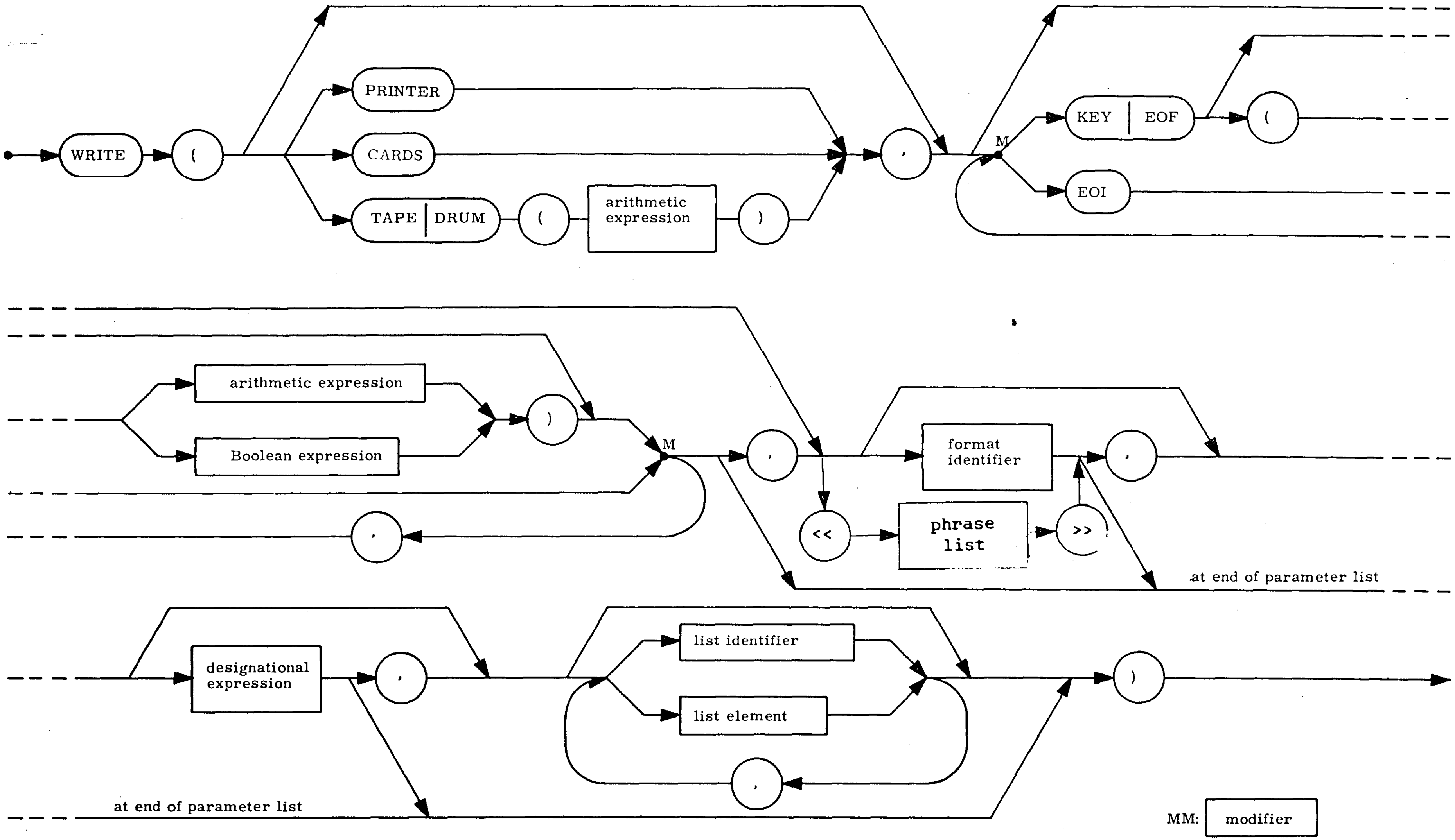


<u>Explanation</u>: The READ statement reads data from the specified input device into the variables indicated by the list elements. The designational expressions are used as exit points in case end-file or end-information conditions are met on that device.

<u>Examples</u>:

```
READ(CARDS,LEOF,LEOI,A,B,C,S,EPSILON) $
READ(DRUM(INDEX), FOR I=(1,1,KMAX) DO FOR J=(1,1,LMAX) DO ERG(I,J)) $
READ(DATE) $
```

output statement ::=

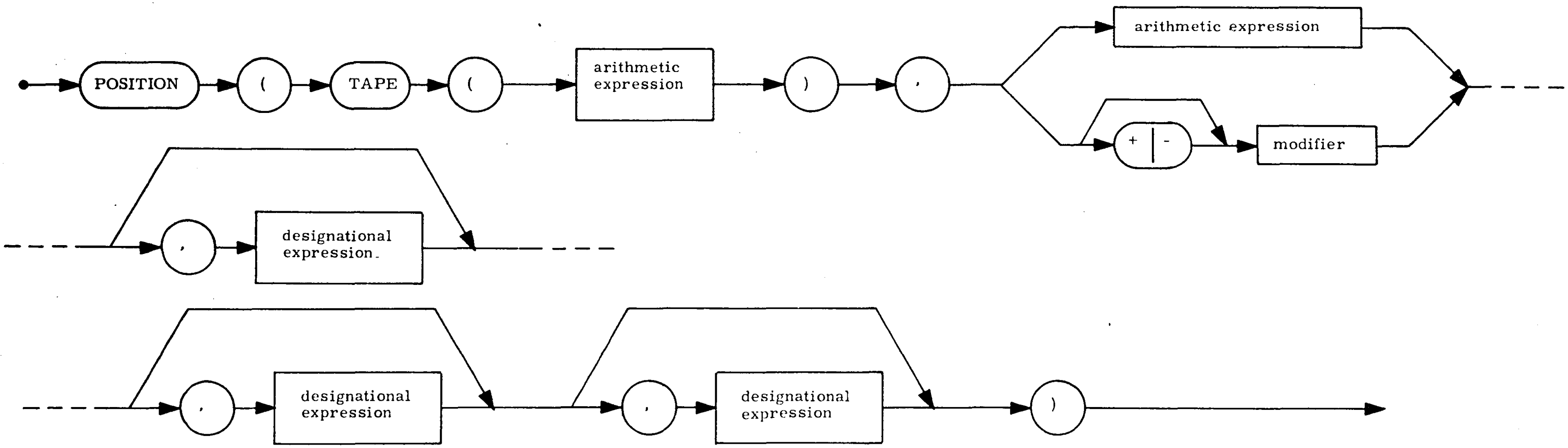


Explanation: The WRITE statement outputs the values defined by the lists to the specified device. Modifiers (KEY,EOF,EOI) produce special marks on tape, a format controls editing on paper and punched cards, the designational expression is used as a return print if the output device functions abnormally.

Examples:

```
WRITE (PRINTER, F10, FOR I=(1, 1, N) DO A(I,J)) $
WRITE ('CHECKPOINT CHARLIE', A) $
WRITE (TAPE(0),KEY(I),ABORTLAB,DUMPLIST) $
WRITE (TAPE(OUTPUT),EOF('LAST'),EOI) $
```

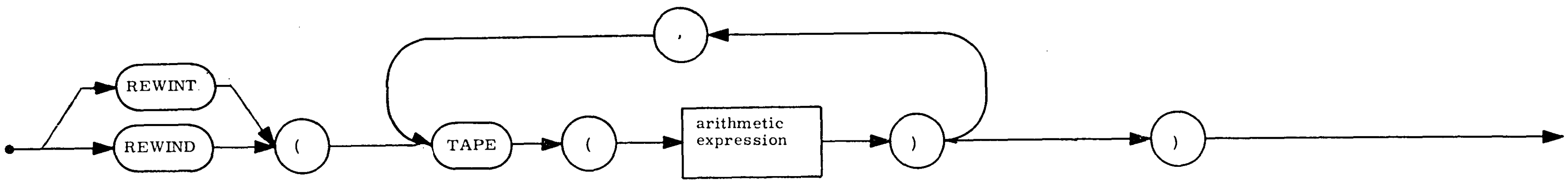
position procedure statement ::=



Explanation: The procedure POSITION is used to position a tape forward or backward a number of records or to search for a KEY, EOF, or EOI marker. The designational expressions are used as exits in cases of search failure.

Examples: POSITION (TAPE(0), -2) $
POSITION (TAPE(INPUT), KEY('PRICES'), ABORT) $
POSITION (TAPE(OUTPUT), EOI) $

rewind statement ::=



Explanation: The REWIND statement will rewind the specified tapes. The REWINT will cause the units to be rewound with interlock (read/write protect).

Examples:

```
REWIND (TAPE (INPUT), TAPE(OUTPUT)) $
REWINT (TAPE(I),TAPE(A),TAPE(J)) $
```