UNIVAC 1100 SERIES NU ALGOL

PROGRAMMER REFERENCE

SPERRY UNIVAC COMPUTER SYSTEMS

A

UP-7884

This document contains the latest information available at the time of publication. However, the Univac Division reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Univac Representative.

UNIVAC is a registered trademark of the Sperry Rand Corporation.

Another trademark of the Sperry Rand Corporation in this publication is:

FASTRAND

PRINTED IN U.S.A.

UPDATING PACKAGE A

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	PAGES NUMBERED
Cover/Disclaimer	†	†
Page Status Summary	N.A.	**
Contents	1 thru 6	1 Rev. A thru 6 Rev. A
Section 1	1 thru 3	1 Rev. A thru 3 Rev. A
Section 2	3 and 4	3 Rev. A and 4 Rev. A
Section 5	13 and 14	13 and 14 Rev. A
Section 7	15 and 16	15 Rev. A and 16 Rev. A
Section 9	1 thru 3	1 Rev. A thru 3 Rev. A
	N.A.	4 Rev. A** thru 10 Rev. A**
Section 10	7 and 8	7 and 8 Rev. A
	N.A.	8a Rev. A**
Appendix A	1 and 2	1 Rev. A and 2 Rev. A
Appendix D	3	3 Rev. A
Appendix F	5 and 6	5 Rev. A and 6 Rev. A
Index	1 thru 5	1 Rev. A thru 5 Rev. A
	N.A.	6 Rev. A**
User Comment Sheet	N.A.	** Add User Comment Sheet following Index

† Destroy old cover and file new cover.
** These are new pages.

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

PAGE STATUS SUMMARY

ISSUE: UP–7884 UPDATE A

Section	Page Number	Update Level
Cover/Disclaimer		A
PSS	1	A
Acknowledgment	1	Orig.
Contents	1 thru 6	A
1	1 thru 3	A
2	1 and 2 3 and 4	Orig. A
3	1 thru 7	Orig.
4	1 thru 15	Orig.
5	1 thru 13 14	Orig. A
6	1 thru 4	Orig.
7	1 thru 14 15, 16 17 thru 38	Orig. A Orig.
8	1 thru 46	Orig.
9	1 thru 10	A
10	1 thru 7 8 and 8a 9 thru 12	Orig. A Orig.
Appendix A	1 and 2	A
Appendix B	1 thru 4	Orig.
Appendix C	1	Orig.
Appendix D	1 and 2 3	Orig. A
Appendix E	1 thru 37	Orig.
Appendix F	1 thru 4 5 and 6	Orig. A
Index	1 thru 6	A
User Comment Sheet		
Total	223 pages and covers	

Section	Page Number	Update Level

Section	Page Number	Update Level

ACKNOWLEDGEMENT

The NU (Norwegian University) ALGOL System was designed and implemented as a joint effort of the Norwegian Computing Center, Oslo, and the Computing Center at the Technical University of Norway, Trondheim.

CONTENTS

TABLES

1. INTRODUCTION

1.1 GENERAL

NU (Norwegian University) ALGOL is a language for communicating scientific and data processing problems to the UNIVAC 1100 Series Systems. The basis for this language is the "Revised Report on the Algorithmic Language, ALGOL 60" (Communications of the ACM, Vol. 6, January 1963, 1-17). This implementation of ALGOL 60 is very close to that of the report. Its one significant difference is the omission of all OWN variables. Some of its more significant additions include three new data types (STRING, COMPLEX, and REAL2). Provision is made for inclusion of procedures written in assembler language of FORTRAN V.

NU ALGOL is compatible with UNIVAC 1106/1108 ALGOL with the few exceptions noted in Appendix D, "UNIVAC 1106/1108 ALGOL and NU ALGOL Differences." The major differences between the two languages are the actual method of compilation, the extended input/output facilities, and a major improvement in both run-time and compile-time security and speed.

This manual has been designed to provide quick reference to all features of the language so that programmers familiar with ALGOL may look up points easily. At the same time, many examples have been inserted to allow inexperienced programmers to become familiar with NU ALGOL.

No attempt has been made to illustrate all possible constructions; however, Appendix E contains a complete syntax chart for NU ALGOL.

Although the ALGOL report previously cited uses underlining to delineate basic symbols, this manual does not. All explanations and examples give the basic symbols as they appear on the 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 unambiguous definition of syntax, see Appendix E.

1.2 THE NU ALGOL COMPILER

The NU ALGOL compiler is a program which accepts statements expressed in ALGOL and produces programs for the UNIVAC 1100 Series Systems.

---

*See UNIVAC 1106/1108 ALGOL Programmer Reference, UP-7544 (current version)

An ALGOL program is a sequence of statements written in ALGOL language. These statements 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 in syntax (that is, for errors in the forms or construction of statements) and reports these errors 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.

1.3 DEVIATIONS FROM ALGOL 60

There are several differences between ALGOL 60, as defined in the revised report, and NU ALGOL. Since ALGOL is intended as a standard language and compatibility of programs between machines is becoming more and more important, those differences must be explicitly pointed out. They fall into two classes: extensions to ALGOL 60 and definition of things left undefined by the report, and modifications or omission of ALGOL 60 entities.

1.3.1 Extensions to ALGOL 60

Extensions to ALGOL 60 include the following:

- The addition of STRING and STRING ARRAY variables has been made to enhance the value of ALGOL as a data processing language.
- The addition of the arithmetic types COMPLEX and REAL2 has been made to enhance the value of ALGOL to scientific users.
- XOR has been added to the list of logical operators.
- EXTERNAL PROCEDURE declarations have been implemented to allow easier programming of large problems and the building of program libraries.
- Input and output routines have been defined along with FORMAT and LIST declarations to be used by the routines.
- A compact form for GO TO and FOR statements has been provided.
- Variables are zeroed upon entry to a block so that initialization statements are not required.
- The controlled variable of a FOR statement has a defined value when the statement is terminated by exhaustion of the FOR list.
- The addition of utility statements to ease debugging and to control various actions of the compiler and runtime system.

1.3.2 Deletions from ALGOL 60

Deletions from ALGOL 60 are as follows:

1. The following limitations have been imposed.
   a. Identifiers are unique only with respect to their first 12 characters.
   b. Identifiers and numbers may not contain blanks.
   c. Certain ALGOL words may only be used in a specific context.
2. OWN variables are excluded.
3. Numeric labels are not allowed.
4. The comma is the only parameter delimiter allowed in a procedure call. ←
5. The result of an integer raised to an integer power is always of type REAL.
6. All the formal parameters of a procedure must be specified.
7. In a Boolean expression, only those operands necessary for determining the result are evaluated.

2. BASIC INFORMATION

2.1 BASIC SYMBOLS

The following symbols have meaning in NU ALGOL.

2.1.1 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

2.1.2 Compound Symbols

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

//	(integer divide)
**	(exponentiation)
&&	base 10 scale factor for double precision constants
:=	assignment (same as =)
..	colon same as :
<<	literal format left bracket
>>	literal format right bracket
''	string quote within a string constant

A set of reserved words such as:

BEGIN END IF THEN

A complete list of reserved words is given in 2.5.1. For details of the character set, see Appendix A.

2.2 IDENTIFIERS

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

The following rules apply to identifiers:

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

The first character of an identifier must be a letter.

Spaces are not allowed within an identifier.

Although up to 72 characters may be used to make an identifier, only the first 12 uniquely specify the identifier.

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

EXAMPLES:

```
A     P060     Z1Z4     KAF1

      NONLINEARRESIDUE
      NONLINEARRESULT
```

The two identifiers in the second example above 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 described in detail in Section 5. Briefly, the following are true:

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

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

3. 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

The source code to the compiler must be input on a line-by-line basis; for instance, from punched cards or a typewriter terminal.

The following rules should be followed:

Only columns 1 through 72 are read for information.

Columns 73 and beyond may be used for any purpose.

One or more statements may be placed on one line or one statement may occupy several lines. A number, identifier, or reserved word may not be broken up to continue on the next line.

2.5 SPECIAL IDENTIFIERS

There are two sets of special identifiers; reserved identifiers and standard identifiers.

2.5.1 Reserved Identifiers

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

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

2.5.2 Standard Identifiers

The following identifiers may be used without explicit declarations:

ABS	DRUMPOS	LISTINPUT	RE
ACARDS	DOUBLE	LN	READ
ACCEPTERRORS	ENTIER	MARGIN	REWIND
ALPHABETIC	EOF	MAX	REWINT
APRINTER	EOI	MIN	SIGN
APUNCH	ERLANG	MOD	SIN
ARCCOS	ERRORDUMP	NEGEXP	SINH
ARCSIN	EXP	NOLINENUMBERS	SQRT
ARCTAN	EXTERNALCOMPILATION	NORMAL	SYMBOLIC
ASSEMBLERLISTING	FILE	NOSUBSCRIPTCHECK	TAN
BLOCKMESSAGE	FILEINDEX	NOSYMBOLIC	TANH
CARDS	GOTOTRACE	NOWARNINGS	TAPE
CBROOT	HISTD	NUMERIC	TIME
CLOCK	HISTO	POISSON	TIMECHART
COMPILEABORT	IFTABLE	POSITION	TIMETRACE
COMPL	IFTRACE	PRINTER	TIMING
CORE	IM	PROCEDURETRACE	UNIFORM
COS	INT	PSNORM	VALUEDUMP
COSH	KEY	PUNCH	VALUETRACE
DISCRETE	LENGTH	PUNCHASSEMBLER	WRITE
DRAW	LINEAR	RANDINT	XQTABORT
DRUM	LINETRACE	RANK	

These identifiers may, however, be redeclared for other use. For details on standard procedures, see 7.4.

3. DECLARATIONS

3.1 GENERAL

An ALGOL program may be broken into logical segments called blocks, which are complete and independent units. Block structure is discussed in Section 6. One important property of a block is that, at the beginning of the block, all local entities that are to be referenced inside the block must be declared. Declarations determine how the compiled program will treat certain of its elements; thus it is necessary to precede the use of an identifier with a declaration of type. An identifier may appear in only one declaration within a block; however, a block may contain blocks within itself (as shown in 6.2). Any of these blocks may declare variables taking on names used in outer blocks, thus redefining variables for the inner block. All identifiers used in a program, except standard procedure identifiers, must be declared.

3.2 TYPE DECLARATIONS

The type declaration defines the type of variable named by an identifier. 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, the values may be string values and truth values, all of which 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	$(-3.37 \times 10^{38}, -1.48 \times 10^{-39})$, 0, $(1.48 \times 10^{-39}, 3.37 \times 10^{38})$ 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.
REAL2		$(-9.0 \times 10^{307}, -3.6 \times 10^{-308})$, 0, $(3.6 \times 10^{-308}, 9.0 \times 10^{307})$ Up to 18 significant digits

<TYPE >	VALUE	LIMITS
STRING	Alphanumeric characters	Any character in the UNIVAC 1100 Series character set.

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.

3.3 DECLARATION OF SIMPLE VARIABLES

A simple variable is a nonsubscripted 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 $
```

The declaration takes the form:

<type><list of identifiers> $

- <type> is defined in 3.2.
- List of identifiers means one identifier (see 2.2) or several identifiers separated by commas.
- The declaration ends with the character $ or ;

3.3.1 Declaration of a Simple String

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

The declaration of a simple string has the form:

STRING <identifier> (<string part>)

- Identifier is defined in 2.2.
- String part is an integer expression (in the outermost 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 3.3.2.)

EXAMPLES:

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

In an inner block also:

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

3.3.2 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 means of a subscripted variable or the call of a type procedure, as these will be taken as a substring declaration. If the type procedure or array and the main string are declared in the same block, this ambiguity will give the error message "DOUBLE DECLARATION."

EXAMPLE:

```
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.

3.3.3 Storage Required by Simple Variables

The storage of the UNIVAC 1000 Series Systems 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.

3.4 DECLARATION OF SUBSCRIPTED VARIABLES (ARRAYS)

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.

The declaration of an array has the form:

<type> ARRAY <array list>$

- <type> is defined in 3.2. If type is omitted, the type REAL is assumed.

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:

<arithmetic expression>: <arithmetic expression>

- Section 4 defines arithmetic expression.

NOTE: In the outermost block, the arithmetic expression can only be a constant.

EXAMPLES:

In an outermost block:

```
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) $
```

3.4.1 Rules for Array Declarations

The rules for array declarations are as follows.

- Each bound pair defines the values the corresponding subscript may take. In NU ALGOL, the number of subscripts is limited to 10.
- 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.
- The arithmetic expressions must be of type INTEGER or of a type which can be converted to INTEGER (REAL,REAL2).

3.4.2 Meaning of Array Declarations

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)
```

The use of a subscripted variable consumes substantially more processor time and program space than the use of a simple variable.

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.

All of these arrays occupy different areas of storage.

3.4.3 Declaration of a String Array

Subscripted STRING variables may be declared using the STRING ARRAY declaration. A string array is an array whose elements are strings. A string array declaration has the form:

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

- An identifier is defined in 2.2.
- The term string part is defined in 3.3.1.
- The term bound pair list is defined in 3.4.

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)$
```

3.4.4 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)

3.5 OTHER DECLARATIONS

The following special declarations are described in the sections listed.

DECLARATION	PARAGRAPH
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 GENERAL

An expression is a rule for computing a value. There are four kinds of expressions: arithmetic, Boolean, string, and designational. Expressions are composed of operands, operators, and parentheses. Operands are constants, variables, function designators, or other expressions. Operators are symbols which designate arithmetic, relational, or logical operations.

- 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

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.

4.2.1 Types of Values

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

INTEGER
REAL
REAL2
COMPLEX

4.2.2 Arithmetic Operands

The operands of arithmetic expressions are constants, variables, type procedures, or other arithmetic expressions.

4.2.2.1 Arithmetic Constants

The type of a constant depends on the form in which it is written. No blanks are allowed in a constant. See 3.2 for the limits of arithmetic constants. The rules given in Table 4-1 apply.

Table 4-1. Rules for Arithmetic Constant Formation

TYPE OF CONSTANT	RULES FOR FORMATION	EXAMPLES
INTEGER	A string of 11 or fewer digits possible preceded by a + or -.	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 18 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
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 x 10^6 or 1000000.0
3.1629&&-4 or 3.1629&-4 means 3.1629x10^{-4} or 0.00031629.

4.2.2.2 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 (i.e., an element of an array).

4.2.2.3 Arithmetic Type Procedures

The declaration of a type procedure is described in 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.

4.2.3 Arithmetic Operators

4.2.3.1 The Operators

The following arithmetic operators are defined in NU ALGOL and have the meanings indicated in Tables 4-2 and 4-3.

Table 4-2. Arithmetic Operator Meaning

OPERATOR	MEANING
+	If not preceded by an operand then unary 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 unary 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.

Table 4-2. Arithmetic Operator Meaning (cont)

OPERATOR	MEANING
*	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 is to be divided by the operand following the operator. Both operands, if necessary, are 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)))

Table 4-3. Arithmetic Operator Examples and Results

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.

4.2.3.2 Precedence of Arithmetic Operators

The precedence of the arithmetic operators is:

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

This means that in a parenthesis-free expression, all exponentiations will be carried out (from left to right), all multiplications and divisions are executed (also from left to right), and 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 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 +

4.2.3.3 Use of Parentheses

It may be useful to group operations by means of parentheses, even when not strictly necessary, so that the intended order of evaluation is immediately visible to the reader of a program.

4.2.4 Type of Arithmetic Expressions

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

- 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

- 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

- 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 compile-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

A Boolean expression is a rule for computing a Boolean value, that is, TRUE or FALSE. In a Boolean expression, only those operands necessary for determining the result are evaluated. A Boolean expression may only produce a value of type BOOLEAN. Boolean constants are written as the character sequences TRUE or FALSE for the appropriate values. Boolean variables are those variables whose identifiers have been declared to have type BOOLEAN. They may be simple or subscripted (i.e., a member of a BOOLEAN array).

The declaration of a type procedure is described in 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

The Boolean operators given in Table 4-4 are defined in NU ALGOL to have the following meanings only if A and B are BOOLEAN expressions.

Table 4-4. 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

The precedence of Boolean operators is as follows:

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 4.2.4 and 4.4).

4.3.2 Relational Operators

Relational operators are defined in NU ALGOL to have the meanings given in Table 4-5. C and D are arithmetic or string expressions.

NOTE: If C or D are of type COMPLEX, only EQL or NEQ may be used.

Table 4-5. 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

For strings, the comparisons are made on a character by character basis, starting with the leftmost character. If the strings are of unequal length, the string of shorter length is considered to be filled with blanks to the longer length. To determine the greater or lesser relations, the characters are ranked by their internal value as shown in Appendix A, Table A-1.

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:

EXPRESSION	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

Arithmetic, Boolean, and relational operators have the following precedences.

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 performed. (See also 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

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. A string quote may be made part of a string constant by the use of a double string quote ('').

EXAMPLES:

```
'NU ALGOL'

'THIS IS A STRING CONSTANT'

'BAD * ? ! / + - WORDS'

'HE SAID:  "YES".'
```

String variables are those variables appearing in a STRING declaration. 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.

4.5.2.1 Arithmetic Operations on Strings

Arithmetic operators may be used between string operands if the string involved contains only digits in the form of INTEGER constants (including sign). If the string is not in the form of an integer constant (containing either 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 $
```

4.5.2.2 Relational Operations on Strings

The equality or collating sequence of strings may be tested using the relational operators (see 4.3.2).

4.5.3 Substrings

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

4.5.3.1 Declared Substring

Substrings may be declared in the declaration of the main string (see 3.3.2).

4.5.3.2 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 in 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 whole string K.

If no start position or length is given, the main string is referenced.

EXAMPLE:

```
STRING K(50)$
```

K and K(1,50) are equivalent

4.5.3.3 Substrings of Members of String Arrays

A reference to a substring of a subscripted string variable is written in 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 3.4.3.

4.6 DESIGNATIONAL EXPRESSIONS

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

- a label

- a switch identifier with an index
- IF < Boolean expression > THEN < simple designational expression> ELSE <designational expression>

  where Boolean expression is described in 4.3. Simple designational expression is either (1) or (2) or (3) enclosed in parentheses.

These expressions have the following meanings:

1. A label refers to that point in the program where the label is declared (see 4.6.1).
2. A switch identifier with an index (e.g.,i) refers to the designational expression in the i^{th} position of the list of designational expressions in the switch declaration (see 4.6.2). If an actual switch index is less than 1 or greater than the number of designational expressions in the list, then the GOTO statement is not executed.
3. 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 is evaluated to the value TRUE, the designational expression is used if the Boolean expression is evaluated to the value FALSE.

4.6.1 Labels

Control may be transferred to a specific program point by the use of a GOTO statement. This program point is called a label. Labels are declared by placing an identifier in front of a statement and separating it from the statement by the colon symbol (:).

EXAMPLE:

```
LAB1 : X = 5$
```

Since NU ALGOL labels are identifiers (see 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

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 > $

- Identifier is as defined in 2.2.
- List of designational expressions is a set of designational expressions separated by commas. Designational expressions are described in 4.6.

EXAMPLES:

```
SWITCH CHOICE = P1, IF A GTR 2 THEN L ELSE Z $

SWITCH JUMP = CHOICE(1), CHOICE(2) $

COMMENT NOTICE THAT A SWITCH IDENTIFIER WITH INDEX IS
A DESIGNATIONAL EXPRESSION $
```

4.7 CONDITIONAL EXPRESSIONS

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. Conditional expressions have the form:

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

- Boolean expression is described in 4.3.

Simple expression is any of the expressions (arithmetic, 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.

Expressions follow these rules:

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.

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 in Table 4-6.

Conditional expressions used as operands must be enclosed by parentheses.

Table 4-6. 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)$

X = IF B THEN X ELSE D $
    Arithmetic expression of type REAL2

LETTERS = IF X GTR Y THEN LETTERS (1,4) ELSE LETTERS (4,8)$
                        String expression

B = IF D LSS E THEN NOT  B  ELSE D LSS E$
                Boolean expression

C = (IF B THEN (IF NOT B THEN X ELSE Y)
     ELSE IF X GTR Y THEN D ELSE E)     + 20$
    Arithmetic expression of type REAL2
```

5. STATEMENTS

5.1 GENERAL

The ALGOL statement is the fundamental unit of operation within the language. The operations to be performed are specified by statements which may be divided into two classes:

- Assignment statements
- Control statements

This section discusses assignment statements (see 5.2), and combination of statements (see 5.3).

The compiler translates successive statements in the order in which they appear in the program. The statements are also executed in this same order unless the programmer interrupts this normal sequence with a "transfer of control." Once the transfer has taken place, successive statement sequencing continues from the new point of reference.

Transfer of control in ALGOL is accomplished through use of three kinds of control statements - unconditional (see 5.4), conditional (see 5.5), and repetitive (see 5.6).

5.2 ASSIGNMENT STATEMENTS

An assignment statement is of the form:

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

where the V_i are variables (either simple or subscripted) and E is an expression. The sign (=) or (:=) means "assign" or "replace."

5.2.1 Rules for Performing Assignment

If V is a subscripted variable, evaluate its subscript expressions, thus determining 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.

5.2.2 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 assignment 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 9.0,Y BE-
                                 COMES 9.0$
```

5.2.3 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 rules given in Table 5-1.

Table 5-1. 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	Converter 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	

5.2.4 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 at the left.

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 'AAAAAAAAAAAAAAA'.$
```

5.3 COMPOUND STATEMENTS

A compound statement is a group of ALGOL statements enclosed by the words BEGIN and END. A compound statement may be used wherever one ALGOL statement is allowed. Compound statements are very useful in conditional and repetitive statements (see 5.5 and 5.6) 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 GO TO STATEMENTS

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 4.6.)

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 $
```

5.5 CONDITIONAL STATEMENTS

Conditional statements may be used to select the next statement depending on the value of a Boolean expression. There are two types of conditional statements, one with alternative and one without. The forms are given below.

5.5.1 Conditional Statement Form Without Alternative

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

Boolean expression is described in 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 $
```

5.5.2 Conditional Statement Form With Alternative

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

- Boolean expression is described in 4.3.
- Unconditional statement is any statement other than a conditional statement, including a compound statement. 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 $
```

5.5.3 Conditional Statement Action Without Alternative

The action of a conditional statement without alternative is given in Table 5-2.

Table 5-2. Conditional Statement Action Without Alternative

BOOLEAN EXPRESSION EVALUATES TO	ACTION
TRUE	Execute unconditional statement after THEN
FALSE	Execute statement after conditional statement

5.5.4 Conditional Statement Action With Alternative

The action of a conditional statement with alternative is given in Table 5-3.

Table 5-3. Conditional Statement Action 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 $
```

EXAMPLES: (cont)

```
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 $
```

5.6 REPETITION STATEMENTS - FOR STATEMENTS

The FOR statement facilitates programming iterative operations. A part of the program is iterative if it is to be executed repeatedly a specified number of times, if it is to be executed for each one of a designated set of values assigned to a variable, or if it is to be executed repeatedly until some condition is fulfilled. The FOR statement handles any of these three conditions.

The FOR statement has the form:

FOR V = < list of FOR list elements > DO < statement > $

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

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

There are three possible kinds of FOR list elements:

- < arithmetic expression >

- < arithmetic expression > STEP < arithmetic expression > UNTIL < arithmetic expression >
- < arithmetic expression > WHILE < Boolean expression >

5.6.1 Simple List Element

The controlled variable V is successively given the values of the arithmetic expressions, e_1, e_2, e_3, - - - e_N, as seen below, and the statement S is executed once for each value of V.

FOR V = < arithmetic expression > DO S $

or

FOR V = e_1, e_2, e_3, e_4, - - - e_N DO S $

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.

EXAMPLE:

Step 1. Evaluate the expression.

Step 2. Assign the value to the controlled variable, converting to the type of the controlled variable if necessary.

Step 3. Execute the statement following DO.

Step 4. If there are no more FOR list elements, then execute the next statement.

Step 5. If there is another FOR list element, repeat from step 1.

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

A has the value 10, B the value 5.

STEP	EXPRESSION		VALUE OF C	VALUE OF TOTAL
	NUMBER	VALUE	0	0
1	1	15		
2			15	
3				15

STEP	EXPRESSION		VALUE OF C	VALUE OF TOTAL
	NUMBER	VALUE	0	0
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			

5.6.2 STEP - UNTIL List Element

In both following 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.

```
FOR V = A STEP B UNTIL C DO S $
```

or

```
FOR V = (A,B,C) DO S $
```

- A is the starting or initial value of V
- B is the increment by which V is increased algebraically
- C is the limiting or terminal value of V

EXAMPLE:

Step 1. Evaluate the expression A; call this value X.

Step 2. Assign the value X to the controlled variable, converting it to the type of the controlled variable if necessary.

Step 3. Evaluate the expressions B and C and convert to the type of the controlled variable if necessary.

Step 4. If the value of B is negative, then go to step 6.

Step 5. If the value of X is greater than the value of C, then go to step 10, otherwise go to step 7.

Step 6. If the value of X is less than the value of C, then go to step 10.

Step 7. Execute the statement after DO.

Step 8. Add the value of X to the value of B - call the result X.

Step 9. Start again at step 2.

Step 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.

```
INTEGER I $ REAL J,K $

INTEGER ARRAY Z(1:4) $

  J = 5.2 $  K = 20.6 $  I = 2 $

FOR Z (I) = J + K STEP - J - I UNTIL - 41

  DO I = I + Z (2) $
```

- In this example the initial value expression A is J + K.
- the step B is - J - I
- 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	5.2	20.6
1	25.8			26				
2					26			
3		-7	-41					
4	Go to step 6							
6	26 > -41 - do next step							
7						28		
8				-7				
9	Go to step 2							
2					-7			
3		-33	-41					
4	Go to step 6							
6	-7 > -41 - do next step							
7						21		
8				-33				
9	Go to step 2							
2					-33			
3		-26	-41					
4	Go to step 6							
6	-33 > -41 - do next step							
7						-12		
8				-26				
9	Go to step 2							
2					-26			
3		7	-41					

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
4	Go to step 5							
5	-26 >	-41 - Go to step 10						
10	No more FOR list elements, go to next statement							

EXAMPLE:

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 $
```

Perform a group of statements N times.

```
INTEGER I,N $   REAL   X,Y $

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 $
```

Set specific 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 $
```

5.6.3 WHILE List

Arithmetic and Boolean expressions used below are as described in Section 4.

FOR V = < arithmetic expression > WHILE < Boolean expression > DO S $

EXAMPLE:

Step 1. Evaluate the arithmetic expression.

Step 2. Assign the value of the arithmetic expression to the controlled variable, V, converting if necessary.

Step 3. Evaluate the Boolean expression.

Step 4. If the Boolean expression has the value FALSE then go to step 7.

Step 5. Execute the statement after DO.

Step 6. Go to step 1.

Step 7. If there are no more FOR list elements, execute the statement after the FOR statement, otherwise take the next FOR list element.

```
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) $
```

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 $
```

EXAMPLE: (cont)

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			

5.6.4 Special Rules for FOR Statements

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.

A GOTO leading to a label within the FOR statement is illegal. A label may, however, be used for a jump within the statement following DO.

5.7 OTHER TYPES OF STATEMENTS

Input/Output statements are described in Section 8.

Procedure statements or calls on procedures which do not have a type are described in Section 7.

Blocks as statements are described in Section 6.

→ The utility statements are described in 9.2.

6. BLOCKS

6.1 GENERAL

The ALGOL block affects a grouping of a set of variables and the statements involving those 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.

A block has the following form:

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

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

6.2 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
```

EXAMPLE: (cont)

```
        REAL E, F $

        E = A * B $    F = A/B $

      END $

        A = A + B $

  END $
```

The blocks in the preceding example 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.

6.3 LOCAL AND GLOBAL IDENTIFIERS

All identifers declared within a block are called local identifers (i.e., local to the given block). Any identifiers that do not occur in declarations in the given block, but appear in a block containing the given block, are called global, or nonlocal (to the given block) identifiers. Each block introduces, at the time it is entered, a new level of nomenclature in the sense that all identifiers declared for the block assume the meaning implied by the declaration.

EXAMPLE:

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

END $
```

Where blocks B2 and B3 are nested in block B1.

- Identifiers that are declared in B1, but not in B2 or B3, are local in B1 and global in B2 and B3.
- Identifiers that are declared in B2 are undefined in B1 and B3. They are local in B2.
- Identifiers declared in B3 are undefined in B1 and B2. They are local in B3.
- 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.

- Upon entering a block, variables are initialized to 0 if arithmetic, to FALSE if Boolean, and to blanks if string.
- 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.

EXAMPLE:

```
BEGIN

  REAL A $

    A = 50.0 $      COMMENT A IS LOCAL AND REAL $

  BEGIN

    INTEGER A $

    A = 5 $         COMMENT A IS LOCAL AND INTEGER $

  END $

  BEGIN

    A = 25.0 $      COMMENT A IS GLOBAL AND REAL $

  END $

END $
```

6.4 LOCAL AND GLOBAL LABELS

Labels are declared, as explained in 4.6.1, 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.

6.5 USE OF BLOCKS

Blocks are used 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.

Blocks are used to save main 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 MAIN STORAGE AREA AS A AND B
  IN THE BLOCK ABOVE $

 END $

END $
```

7. PROCEDURES AND TYPE PROCEDURES

7.1 PROCEDURES

The ALGOL procedure provides a convenient means of defining an algorithm and giving it a name so that it may be referenced or called anywhere within the scope of the declaration of the procedure identifier. Furthermore, different actual parameters or arguments may be passed to the procedure at each call.

7.1.1 Procedure Declaration

The procedure declaration consists of the procedure heading and the procedure body. The identifier of the procedure appears in the procedure heading, followed by a list of names which designate formal parameters. The formal parameter list may be empty, but if it is not, each formal parameter name must be further defined by the specification part. The procedure declaration has the form:

Procedure heading:

PROCEDURE identifier (formal parameter list) $

<value part>$

<specification part>$

Procedure body:

<statement>$

value part is described in 7.1.3

identifier is as described in 2.2

formal parameter is described in 7.1.2

specification part is described in 7.1.1.2

7.1.1.1 Identifiers in the Procedure Body

The statement which is the procedure body may be a block. Identifiers declared in the block are local to the block. (See 6.2.) Identifiers declared in the block containing the procedure declaration are global to the procedure and may be referenced by statements in the procedure body.

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 $
```

7.1.1.2 Specification Part

The specification part gives the type and kind of the formal parameters, and may also indicate the modes of transmission of the actual parameters. The form of a specification is:

<specifier><list of identifiers>$

list of identifiers has the usual meaning, except that in this case the identifiers may only be formal parameters.

Table 7-1 gives the possible specifiers.

Table 7-1. Specifiers and Parameters

USE THE SPECIFIER	WHEN A FORMAL PARAMETER IS TO BE
INTEGER REAL REAL2 COMPLEX BOOLEAN STRING	A simple variable of the specified type

Table 7-1. Specifiers and Parameters (cont)

USE THE SPECIFIER	WHEN A FORMAL PARAMETER IS TO BE
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 7.1.3

NOTE: The value part must come before the specifications.

7.1.1.3 Procedure Body

The procedure body must be only one statement. This statement may be a compound statement or a block. A formal parameter used on the left hand side of an assignment statement must have a variable for actual parameter, unless the format parameter has appeared in the value part.

EXAMPLE OF PROCEDURE DECLARATION:

```
PROCEDURE EXAMPLE (A,B,ANS,C)$
  VALUE B $              COMMENT VALUE PART $
    REAL ARRAY B $       COMMENT OTHER SPECIFICATIONS $
    INTEGER A $
```

EXAMPLE OF PROCEDURE DECLARATION: (cont)

```
    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 $
```

7.1.2 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.

7.1.3 Value Part

The value part causes the value or values of the actual parameter to be copied into a temporary area. These values can then be manipulated or changed without destroying the values of the actual parameter. The form of the value part is:

VALUE <identifier list> $

A main advantage of the value part is that if the actual parameters are expressions, they are evaluated only once. The main implications of this can be seen in 7.1.5.2.

The following kinds of formal parameters may not be placed in a value part:

LABEL, SWITCH, FORMAT, PROCEDURE, LIST

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 $
```

EXAMPLE: (cont)

```
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 $
```

7.1.4 Comments in a Procedure Heading

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:

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


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

7.1.5 Procedure Statement

A procedure statement calls for the sequential execution of a previously defined procedure body. The procedure identifier designates the particular procedure body to be executed and the actual parameter part supplies the arguments to be passed to the procedure.

A procedure statement has the form:

<identifier>(<actual parameter list>) $

identifier is the identifier of the wanted procedure

actual parameter list is a list of variables or expressions

7.1.5.1 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 rules given in Table 7-2 apply.

Table 7-2. Actual and Formal Parameter Correspondence

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.

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 subscripted variable, not an expression or a constant.

A formal parameter whose actual parameter is a constant or an expression may be used for temporary storage if the formal parameter is included in the value part. 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:

For the procedure declared in 7.1.1.3.

```
REAL ARRAY ARY(1:25) $ INTEGER RESULT $

EXAMPLE (15,ARY,RESULT,L1) $

L1:
```

For the procedure declared in 7.1.3.

```
INTEGER K,SIZE $

K = 25 $ COUNT (K,SIZE) $
```

7.1.5.2 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:

- All formal parameters which have not been included in the value part (name parameters), are treated as if they were textually replaced by the corresponding actual parameters in the procedure body. Parameters are re-evaluated each time they are referenced within the procedure body.
- Formal parameters which have been included in the value part are evaluated, and these values are assigned to the formal parameters, which are then used in the procedure body. The corresponding actual parameters are inaccessible to the procedure.

EXAMPLES:

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) $
```

EXAMPLES: (cont)

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 PART $

 END $
```

7.1.6 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

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. In addition to all of the rules for procedures stated in 7.1.1, a few additional rules apply.

7.2.1 Type Procedure Declaration

The type procedure declaration has the form:

```
<type> PROCEDURE<identifier>(<formal parameter list>) $
 <value part> $
<specifications> $
<statements> $
```

- <type> is described in 3.2
- identifier is described in 2.2
- formal parameter list, value part are described in 7.1.3

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

7.2.2 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>)
```

Refer to 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 7.1.6.)

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

EXAMPLES:

```
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

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.

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.

7.3.1 External Declaration

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.

The external declaration has the form:

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

- <type> is as defined in 3.2.
- If no type is given, then the external procedure is a pure procedure as described in 7.1.
- <kind> can be <empty>, ALGOL, FORTRAN, ASSEMBLER, 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.
- ASSEMBLER and LIBRARY means the external procedure is written in the assembler language.

The following descriptions require an adequate knowledge of the UNIVAC 1100 Series Operating Systems, FORTRAN, and assembler language.

7.3.2 ALGOL External Procedures

An ALGOL procedure declaration (see Section 3) may be compiled separately if an E option (see 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 twelve characters of the procedure name marked as an entry point. Such a procedure may be referenced from another ALGOL program as an external procedure if the appropriate declaration and identifier are used.

EXAMPLES:

1. The externally compiled procedure.

```
∇ ALG,EIS <name>

      PROCEDURE RESIDUES (X,Y)$

      VALUE X,Y$ REAL X,Y$

      BEGIN

        .
        .

      END$
```

The main program

```
∇ ALG,IS <main name>

      BEGIN

      EXTERNAL PROCEDURE RESIDUES$

      REAL A,B$

        .
        .

      RESIDUES (A,B)$

        .
        .

END$
```

2. The externally compiled procedure.

```
∇ ALG,EIS <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,IS <main name >

      BEGIN

      REAL ARRAY MATRIX (1:10,1:10)$

      EXTERNAL REAL PROCEDURE DET$

       .
       .
       .

      WRITE(DET(MATRIX,10))$

       .
       .

      END OF MAIN PROGRAM$
```

7.3.3 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>
```

- type is described in 3.2
- identifier list described in 2.2

Actual parameters in calls on such FORTRAN subprograms may be either expressions, arrays or labels. Procedures, 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 occur.

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,IS <name1>
       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,IS <name2>
       BEGIN
       ARRAY MATRIX (1:10,1:10)$
       EXTERNAL FORTRAN REAL PROCEDURE DET$
         .
         .
```

EXAMPLE: (cont)

```
WRITE (DET(MATRIX,10))$

END OF MAIN PROGRAM$
```

7.3.4 Assembler Language Procedures

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

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

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 twelve characters of the name in the identifier list of the EXTERNAL PROCEDURE declaration must be the first twelve 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.

The following listing shows a comparison of ASSEMBLER and LIBRARY procedures.

	ASSEMBLER	LIBRARY
1. Method of parameter transmission	By means of parameter descriptors in main storage	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 ASSEMBLER procedure.	Full checking is done at compile-time.
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.

	ASSEMBLER	LIBRARY
4. Flexibility	Complete information available at run-time about the parameters. The number of actual parameters may vary from one call to another.	Less flexible because allowable actual parameters are determined at compile-time. The number of actual parameters must be equal to the number of formal ones.
5. Example		
Declaration:	EXTERNAL ASSEMBLER 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

7.3.4.1 External ASSEMBLER Procedure

The external ASSEMBLER procedure has the form:

EXTERNAL ASSEMBLER <type> PROCEDURE <identifier list> $

EXAMPLES:

```
EXTERNAL ASSEMBLER PROCEDURE BIT, PACK $

EXTERNAL ASSEMBLER COMPLEX PROCEDURE ARRAYSUM$
```

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

```
F5   FORM   18,6,12

F1   FORM   6,6,6,18

     LMJ    X11,<procedure name>

     F5     <not used>,<type of procedure>,<number of parameters>

     F1     <type>,<kind>,<base register>,<relative data address>
```

- F1 is the parameter descriptor; there is one descriptor 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.

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>.

The return point for a call with N parameters is the contents of register X11 + N + 1.

EXAMPLE:

Call: BIT (X,Y,Z,D,E,F)$

Return: J 7,X11

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,X11

If C is a label exit to C is J *3,X11

See Sections 7.3.4.3, 7.3.4.4 and 7.3.4.5 for description of STRING, ARRAY, and STRING ARRAY parameters respectively.

Assembler language program example:

```
▽ASM,SI < name1 >
  • THE FOLLOWING PROGRAM HAS NO PURPOSE
  • OTHER THAN TO ILLUSTRATE THE ABOVE NOTES
```

```
$(1)                EQUIV           SET UP MNEMONICS
ESP*                HAS THE CALL    ESP (INT,STRING,EXIT LABEL)$
        L,T1        A1,1,X11.       PICK UP TYPE AND KIND
        TE,U        A1,0101.        IF NOT SIMPLE
        J           *3,X11.         INTEGER GO TO ERROR EXIT
        L           A0,*1,X11.      PICK UP VALUE OF INTEGER
        TG,U        A0,1024.        IF THE INTEGER GEQ 1024
        J           *3,X11.         THEN GO TO ERROR EXIT
        L,T1        A1,2,X11.       PICK UP TYPE/KIND FOR SECOND PARAMETER
        TE,U        A1,0701.        IF NOT SIMPLE STRING
        J           *3,X11.         THEN GO TO ERROR EXIT
        L,H2        A1,*2,X11.      PICK UP ADDRESS FROM STRING DESCRIPTOR
        L           A5,1,A1.        PICK UP SECOND WORD OF STRING
        J           4,X11.          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,U         A0,*1,X11.      GIVES ADA
        L           A3,*3,X11.      PICK UP COLUMN
        MSI,H1      A3,1,A0.        MULTIPLY BY D2
        A           A3.*2,X11.      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
ER          ERR$
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,X11.      STORE RESULT IN
J           5,X11.          FOURTH PARAMETER AND RETURN
END.
```

Main program example:

```
∇ ALG,IS < name2 >
 BEGIN
    EXTERNAL ASSEMBLER INTEGER PROCEDURE ESP$
    EXTERNAL ASSEMBLER 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$
```

7.3.4.2 External LIBRARY Procedure

In order to make possible the compile-time checking of 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 parameter list>)$
<value part>
<specification part>$
```

EXAMPLE:

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

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.

Table 7-3 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.

Return from a LIBRARY procedure is always to 0,X11.

EXAMPLE:

```
∇ ALG,IS MAIN
 BEGIN
        COMMENT THIS EXAMPLE SHOWS HOW TO PACK THREE INTEGER
        NUMBERS INTO ONE 1100 SERIES PROCESSOR WORD IN ORDER TO
        SAVE CORE SPACE, AND THEN UNPACK THEM AGAIN FOR COMPUTATION.
        FOR SUCH PACKING THE NUMBERS MUST HAVE ABSOLUTE VALUES LESS
        THAN 2047.
```

EXAMPLE: (cont)

```
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,X2
    L   A2,J,X2
    L   A3,K,X2
    LMJ X11,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,X2
              L,U  A2,J,X2
              L,U  A3,K,X2
              LMJ  X11,UNPACK;
          N = I + J * K;
          UNPACK(NUMBERS(10000-M),I,J,K);
          N = N * K // I + J;
          WRITE(N);
        END;
END MAIN PROGRAM;
▽ASM,S1 PUNP
 PACK*
          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,X11        RETURN TO MAIN PROGRAM
 UNPACK*
          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,X11        RETURN TO MAIN PROGRAM
          END.
```

Table 7-3. Formal and Actual Parameter Combinations

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						String descriptor (see 7.3.4.3)		
String not by value						Address of the string descriptor (see 7.3.4.3)		
Array							Address of the array descriptor (see 7.3.4.4)	
Label								Program address

7.3.4.3 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>
```

- <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.

7.3.4.4 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
ADA+5	D10		dimensions 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:

$$<\text{absolute address of array element } (S_1,S_2 \ldots S_n)>=$$

$$(..((S_n * D_n + S_{n-1}) * D_{n-1} + S_{n-2}) * D_{n-2} \ldots) * D_2 + S_1 + BA$$

For COMPLEX or REAL2 arrays the algorithm has the form:

<absolute address of double array element $(S_1,S_2,\ldots S_n)$>

$$(2*\left[(..((S_n*D_n+S_{n-1})*D_{n-1}+S_{n-2})*D_{n-2}\ldots)*D_2+S_1\right]+BA$$

EXAMPLE:

The array element A(I,J,K) has the address:

(K*D3+J)*D2+I+BA.

The checkword at location FA has the following format:

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

7.3.4.5 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	Same as words ADA through ADA+5 for ordinary arrays
ADA+2	
ADA+3	
ADA+4	
ADA+5	
ADA+6	

The relative string descriptor has the following form:

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

- <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>
```

- <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 ASSEMBLER PROCEDURE XYZ$

  .
  .
  .
XYZ(S1,S2,S3)$
```

7.3.4.6 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

	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 = Start address

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

7.4 STANDARD PROCEDURES

7.4.1 Available Procedures

The procedures given in Table 7-4 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.

Table 7-4. Available Procedures

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
ABS	1	INTEGER	The absolute value of the parameter.	INTEGER
		REAL		REAL
		REAL2		REAL2
		COMPLEX		REAL
ACARDS	1	STRING INTEGER	To direct I/O from or to an alternate card file (see 8.3.6.3).	
ALPHABETIC	1	STRING	TRUE if the string consists only of spaces or alphabetics (A-Z), FALSE otherwise.	BOOLEAN
APRINTER	1	STRING INTEGER	Output to alternate print file (see 8.3.6.3).	
APUNCH	1	STRING INTEGER	Output to alternate card file (see 8.3.6.3).	
ARCCOS	1	INTEGER REAL	arccos (X)	REAL
		REAL2	arccos (X)	REAL2
ARCSIN	1	INTEGER REAL	arcsin (X)	REAL
		REAL2	arcsin (X)	REAL2
ARCTAN	1	INTEGER REAL	arctan (X)	REAL
		REAL2	arctan (X)	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 card punch (see 8.3.4).	

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR 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
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	A complex number with the real part equal to X and the imaginary part equal to Y.	COMPLEX
		(2) INTEGER, REAL, REAL2	Example: COMPL(1,2) gives the complex number <1.0,2.0>.	
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	Drawing from a discrete (cumulative) distribution function (for full description see 7.4.2).	INTEGER
		(2) INTEGER		

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
DRAW	2	(1) REAL (2) INTEGER	TRUE with the probability X, FALSE with the probability 1-X (see 7.4.2).	BOOLEAN
DRUM	0 or 1	INTEGER	Gives input/output routine access to relative address X of random drum. If X is not specified then the next relative address available is used (see 8.3.6).	
DRUMPOS	0	-	Gives next relative drum address (see 8.3.6.2).	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 8.4.5). Only the first six characters of the string are used.	
EOI	0	-	Used by WRITE and POSITION (see 8.4.6).	-
ERLANG	3	(1) REAL (2) REAL (3) INTEGER	A drawing from the Erlang distribution with mean 1/X and standard deviation $1/X\sqrt{Y}$ (for full description see 7.4.2).	REAL
EXP	1	INTEGER REAL	exp (X)	REAL
		REAL2	exp (X)	REAL2
		COMPLEX	exp (X)	COMPLEX

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
FILE	1 or 2	1. STRING INTEGER 2. INTEGER	Directing I/O from or to a specified file (see 8.3.6).	-
FILEINDEX	1	STRING INTEGER	Next relative INTEGER address of an indexed file (see 8.3.6.2).	-
HISTD	2	(1) REAL ARRAY (2) INTEGER	A drawing from a histogram (for full description see 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) using the weight given as the fourth parameter (for full description see 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 8.4.4). Only the first six characters of the string are used.	

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
LENGTH	1	STRING	Number of characters in the string including blanks. Example: STRING S(42)$ LENGTH (S) has the value 42	INTEGER
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 7.4.2).	REAL
LN	1	INTEGER REAL	In (X)	REAL
		REAL2	In (X)	REAL2
		COMPLEX	In (X)	COMPLEX
MARGIN	1	STRING	To change the vertical dimensions on a printer page (see 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 (see 8.7.3).	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 (see 8.7.3).	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)*Y is computed.	INTEGER

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
		(2) INTEGER REAL REAL2	Example: Value of MOD(-48,5) is 2	
NEGEXP	2	(1) REAL (2) INTEGER	A drawing from the negative exponential distribution with mean 1/X (for full description see 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 7.4.2).	REAL
NUMERIC	1	STRING	TRUE if string has the form of an integer, FALSE otherwise (see 4.2.3).	BOOLEAN
POISSON	2	(1) REAL (2) INTEGER	A drawing from the Poisson distribution (see 7.4.2).	INTEGER
POSITION	special list	-	To position a tape (see 8.8.3).	-
PRINTER	0	-	To assign the printer as device to the WRITE statement (see 8.3.5).	-
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 7.4.2).	REAL
PUNCH	0	-	Same as CARDS on output (see 8.3.4).	-

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
RANK	1	STRING	The Fieldata equivalent of the first character of the string. Example: STRING S(12)$ S='DEVICE'$ 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 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	Special list	-	To rewind a file (see 8.8.4).	-
REWINT	Special list	-	To rewind a file and lock if it is a tape (see 8.8.4).	-
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
SIN	1	INTEGER, REAL	sin (X)	REAL
		REAL2	sin (X)	REAL2

Table 7-4. Available Procedures (cont)

NAME	NUMBER OF PARAMETERS	TYPES OF PARAMETERS	RESULT OR USE	TYPE OF RESULT
		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
TIME	0	-	Net CPU-time in 0.1 msec. elapsed since previous call of TIME	INTEGER
UNIFORM	3	1. REAL 2. REAL 3. INTEGER	The value is uniformly distributed in the interval [X,Y] (see 7.4.2).	REAL
WRITE	Special list	-	To send output to a specified device (see 8.8.2).	-

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.

7.4.2.1 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, e.g. U, by a new value according to the following algorithm.

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

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.) For UNIVAC 1100 Series, 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, ... n is called a stream of pseudo-random numbers, and u_i (i = 1,2, ... n) 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 $1 - u_1$, $1 - u_2$, ... n are obtained. In certain situations it can be proved that means obtained from samples based on antithetic 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.

7.4.2.2 Random Drawing Procedures

The following procedures all perform a random drawing of some kind. Unless otherwise explicitly stated, the drawing is affected 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]. 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,

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

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

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

Both a and b must be greater than zero.

9. INTEGER PROCEDURE DISCRETE (A, U); ARRAY A; INTEGER 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 real.

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. REAL PROCEDURE LINEAR (A, B, U); ARRAY A, B; INTEGER 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 real arrays of the same length, that the first and last elements of A are equal to 0 and 1 respective and that A (i) ≥ A (j) and B (i) > B (j) for i > j.

11. INTEGER PROCEDURE HISTD (A, U); ARRAY A; INTEGER U;
The value is an integer in the range [lsb, usb], where lsb and usb are the lower and upper subscript bounds 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 Transfer 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. Transfer functions, given in Table 7-5, are not evoked automatically when the formal and actual types for array identifiers are not the same.

Table 7-5. Transfer Functions

TYPE OF VARIABLE	TRANSFERRED TO TYPE	FUNCTION USED
INTEGER	REAL	Implicit
	REAL2	DOUBLE(X) or Implicit
	STRING	Implicit
	COMPLEX	COMPL(X,0) or Implicit
REAL	INTEGER	INT(X) or Implicit
	REAL2	DOUBLE(X) or 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 GENERAL

Input and output operations are accomplished in UNIVAC 1100 NU ALGOL by means of calls to library procedures. The procedures, READ and WRITE, are more flexible than ordinary procedures written in ALGOL because the number of parameters in an actual call or even the order of the parameters is not rigidly specified. The general form of I/O call is:

< I/O procedure> (< device > , < format > , < modifier list > , < input/output list > , < label list >)

- < I/O procedure > is READ, WRITE, or one of the file-handling procedures (POSITION, REWIND, REWINT);
- <device > specifies the external medium;
- <format> is the name of the format specifying output editing or card layout for input;
- <modifier list > specifies parameters whose action is to output markers in the information which later may be used for positioning;
- < input/output list > is a list of I/O variables and expressions;
- < label list> specifies where control will be transferred in case of contingencies.

This section is organized so that the parameters < device > , < modifier list >, < label list >, < format> and <input/output list > are described in separate paragraphs. 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 (FILE ('A'),EOF('ABC'),LABL1,ARRY)$

READ (CARDS,FORM1,LABL2,LABL2,X,Y,Z)$
```

EXAMPLE: (cont)

```
READ (CARDS,X,Y,FILE('B'),ARRY)$

COMMENT MORE THAN ONE DEVICE ALLOWED$
```

The available input/output procedures are:

PROCEDURE	PARAGRAPH	
READ	8.8.1	
WRITE	8.8.2	
POSITION	8.8.3	Classed as FILE operations
REWIND	8.8.4	
REWINT	8.8.4	

8.2 PARAMETERS TO INPUT/OUTPUT PROCEDURES

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 8.8.

EXAMPLE:

```
FORMAT F(10112,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$
```

In general, all statements should have their parameters in the order given by the form in 8.1. If this order is not observed, the following rules hold.

- Labels may come anywhere and need not be together. However, their order is important. (See 8.5, label list.)
- If device is not before the input/output list, then the device is assumed to be implied device. (See 8.3.3, implied device.)
- The insertion of more device parameters in an I/O call changes the device dynamically, the new device applying to the parameter items which follow.

EXAMPLE:

```
ARRAY A(0:500)$

WRITE (A,FILE ('B'),A)$

COMMENT WILL WRITE ARRAY A ON THE PRINTER AND ON THE SEQUENTIAL FILE
ASSIGNED AS B $
```

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

EXAMPLE:

```
ARRAY A(0:500),B(0:300)$

WRITE(FILE('B'),KEY('A'),A)$

WRITE(FILE('B'),EOF('A'),KEY('B'),B,EOI)$

COMMENT THE FILE 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$
```

- 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)$
```

EXAMPLE: (cont)

```
COMMENT WILL PRODUCE THE FOLLOWING PRINT LINES$

        123          456          789

123.00000     456.00000     789.00000
```

- Formats must come after the device to which they apply.
- 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 list of possible devices follows.

DEVICE	PARAGRAPH
(implied)	8.3.3
CARDS	8.3.4
PUNCH	8.3.4
PRINTER	8.3.5
FILE	8.3.6
DRUM	8.3.6.2
ACARDS	8.3.6.3
APRINTER	8.3.6.3
APUNCH	8.3.6.3
CORE	8.3.7

8.3.2 Actual Devices

The list of actual devices follows.

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
PUNCH	Not allowed	Card punch	
PRINTER	Not allowed	Line printer	Not allowed
FILE	Tape unit, drum or FASTRAND file specified	Tape unit, drum or FASTRAND file specified	Tape unit, drum or FASTRAND file specified
DRUM	FASTRAND or drum file	FASTRAND or drum file	Not allowed
CORE	The string which is parameter	The string which is parameter	Not allowed
ACARDS	Symbiont file	Symbiont file	Not allowed
APRINTER	Not allowed	Symbiont file	Not allowed
APUNCH	Not allowed	Symbiont file	Not allowed

EXAMPLES:

```
INTEGER I$

READ (CARDS,I)$

READ(I)$   COMMENT ARE THE SAME$
```

8.3.3 Implied Devices

Implied devices are used for reading cards or printing. The device parameter is left out for implied devices. READ produces the same action as for device CARDS. WRITE produces the same action as for device PRINTER.

Implied devices have the following restrictions:

- Cannot be used with FILE operations.
- On input only 80 columns may be read from a card.
- On output only 132 columns may be printed.

EXAMPLE:

```
INTEGER A,B,C$

FORMAT F1(A,3(I12,X10))$

READ (F1,A,B,C)$

COMMENT WILL READ CARDS$
```

8.3.4 Devices CARDS and PUNCH

The devices CARDS and PUNCH are used for reading or punching cards (PUNCH is only allowed with WRITE). The card reader (CARDS) 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.

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)$

ENDS
```

Data card form is:

```
12345678910111213141516 17
```

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

The card punch (CARDS or 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$
```

CARDS and PUNCH have the following restrictions:

- They cannot be used with the FILE operations.
- On both input and output there is a maximum length of 80 columns.

8.3.5 Device PRINTER

The device PRINTER is used for printing on a printer. The line printer (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$
```

PRINTER has the following restrictions:

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

EXAMPLE:

```
ARRAY A(-5:6)$

INTEGER I$

FORMAT F1(12(I11,X1),A1)$

WRITE (PRINTER,F1,FOR I=(-5,1,6) DO A(I))$
```

8.3.6 Devices for File Handling

8.3.6.1 Sequential Files

A sequential file can be magnetic tape or simulated on random access storage. It has the form:

FILE (<filename>)[1]

<filename>, if integer, is converted to string. The twelve first characters of the string are taken to be the internal name of the file. If the string is shorter than twelve characters it is space filled to the right. If a non-existent file is referenced, a temporary FASTRAND mass storage file is assigned automatically.

EXAMPLES:

```
ARRAY B,C(1:1000)$

WRITE (FILE('DATA'),B)$

REWIND (FILE('DATA'))$

READ (FILE('DATA'),C)$

WRITE (FILE(1),B)$
```

Use the specified sequential file for input or output (READ or WRITE). Both input and output are double buffered. If the file is on tape it may consist of more than one physical reel. Transition to the next reel is automatic.

[1] The form TAPE (<filename>) is also implemented for sequential files to provide compatibility with EXEC II NU ALGOL (see Appendix F).

EXAMPLE:

```
REAL2 ARRAY D(0:400)$ INTEGER I$

READ (FILE(20),FOR I=(1,1,320) DO D(I))$

WRITE (FILE('A'),FOR I=(1,1,300) DO D(I))$
```

The action with REWINT is as follows:

- If the filename refers to a magnetic tape then this tape is rewound and released so that it can no longer be used. The buffers are released.
- If the filename refers to a sequential drum or FASTRAND file, then the current position of this file is reset to the starting position.

The action with REWIND is as follows:

- For magnetic tapes, the tape is rewound but not released so that it may be used again. The buffers are released.
- The action for sequential drum or FASTRAND files is the same as for REWINT.

EXAMPLE:

```
BOOLEAN DRUMORTAPE$

DRUMORTAPE=TRUE$

REWIND (FILE(IF DRUMORTAPE THEN 0 ELSE 6))$

COMMENT WILL REWIND TAPE ASSIGNED AS A, OR THE FILE HAVING THE NAME 0$
```

The action with POSITION follows:

- The specified sequential drum file is to be used by POSITION. The file will then be searched according to certain parameters. This operation is described in 8.8.3.

EXAMPLE:

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

Restriction: If the file consists of more than one reel of tape, it is not possible to position backwards to a preceding reel.

Sequential files have the following restrictions:

- Sequential files on random access storage can only be accessed in a serial manner. If random access is required it must be done as described in 8.3.6.2.
- FILE does not allow READ or WRITE to use a format. To write formatted output use WRITE (CORE(S),...) and then output the resulting string.

- The input list (see 8.7) 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.

8.3.6.2 Indexed Files

Indexed files are random access files on drum or FASTRAND. Each file has an associated file index or address which is set to indicate where an I/O operation is to be performed. After the I/O operation the file index will be updated to point to the position following the one last used. The file index is initialized to zero which is the first position of the file. Indexed files have the form:

FILE (< filename > , < file index >)

DRUM (< file index >) or DRUM

- < filename > is explained in 8.3.6.1.
- < file index > is an arithmetic expression which is rounded to integer if necessary. If the file is on FASTRAND, the file index is truncated to a multiple of 28 which is the FASTRAND sector length.
- DRUM without an explicit file index means that the current index is to be used.
- DRUM refers to a temporary file of 20,000 words in a word-addressable drum which is automatically assigned.

EXAMPLE:

```
REAL X,Y,Z;
INTEGER I;
I = 56;
WRITE(FILE('DATA',I),X,Y,Z);
COMMENT WILL WRITE THE VALUES OF THE VARIABLES X,Y,Z INTO POSITIONS 56,57
       AND 58 OF THE FILE 'DATA';
WRITE(DRUM(I),X,Y,Z);
COMMENT WRITE THE VALUES INTO THE SAME POSITIONS OF THE FILE PROVIDED FOR
       DEVICE DRUM;
WRITE(DRUM,X,Y,Z);
COMMENT WRITE THE VALUES INTO POSITIONS 59,60 AND 61;
```

The file index is obtained by means of the integer procedures FILEINDEX (< filename >) or DRUMPOS. If < file name> refers to a FASTRAND file the file index returned will point to the first word of the next sector. If the file is non-existent the file index will be zero.

EXAMPLE:

```
INTEGER I;
ARRAY A,B(1:20);
COMMENT 'F1' IS A FASTRAND FILE;
WRITE(FILE('F1',0),A);
WRITE(FILE('F1',FILEINDEX('F1')),B);
I=FILEINDEX('F1');
COMMENT A GOES INTO POSITIONS 0-19 AND B INTO 28-47 THE VALUE OF I IS 56;
READ(DRUM(100),A);
I=DRUMPOS; COMMENT I IS 120;
```

The action with WRITE means the values of the variables of the output list are transferred to consecutive positions in the file starting at the position specified by the file index.

The action with READ means the values of the consecutive positions in the file, starting with the position specified by the file index, are transferred to the input list variables. The current file index after a READ or WRITE may be computed by means of the lengths given in 3.3.3. One exception is that strings will occupy one word less on a file than in core storage.

POSITION sets the file index to the specified position.

EXAMPLE:

```
POSITION(FILE('DATA',30));
COMMENT IF 'DATA' IS ON FASTRAND THE FILEINDEX IS SET TO 28
        OTHERWISE TO 30;
POSITION(DRUM(500));
```

REWIND and REWINT sets the file index to zero and the buffer areas in core are released.

8.3.6.3 Alternate Symbiont Files

Alternate symbiont files are used to read from or write on card, punch, or print files other than the standard ones. They have the form:

ACARDS(< filename >)

APRINTER(< filename >) } only allowed with WRITE

APUNCH(< filename >) }

From the point of view of the program, these devices behave like the standard CARDS, PRINTER, and PUNCH, as described in 8.3.4 and 8.3.5. < filename > is explained in 8.3.6.1. The files are sequential with no possibilities for positioning or rewinding.

READ only allows device ACARDS (< filename >). It will read any file in System Data File (SDF) format. The file may, for instance, have been prepared by means of a WRITE(ACARDS(--)---) statement or by the DATA processor.

WRITE writes a file in card or printer SDF format. The files may then later on be output on the appropriate equipment by means of executive control statements. If the file is temporary, output is done automatically at the end of the run.

8.3.7 Device CORE

The device CORE allows editing to and from a string without using an external device. It has the form:

CORE (<string expression >)

WRITE edits the output list 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$
```

READ edits the string 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 (CORE(S),F,R,I)$
  COMMENT R NOW HAS THE VALUE 1234.56784 AND I HAS THE VALUE 21$
END$
```

CORE has the following restrictions:

- CORE cannot be used with the FILE operations.
- On input (READ) only 80 characters may be edited.
- On output (WRITE) only 132 characters may be edited.
- 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$
```

- Note that nothing is transferred to or from the string until the activate (A) phrase is reached in the format specified.
- If no format is specified, the rules for free format (see 8.6.1) are applied.

8.4 MODIFIER LIST

The modifier list contains directions as to the type of markers to be used on sequential files.

8.4.1 Possible Modifiers

The list of possible modifiers follows.

MODIFIER	PARAGRAPH
EOF	8.4.5
EOF (< parameter >)	8.4.5
-EOF	8.4.5
-EOF (< parameter >)	8.4.5
KEY	8.4.4
KEY (< parameter >)	8.4.4
-KEY	8.4.4
-KEY (< parameter >)	8.4.4
EOI	8.4.6
-EOI	8.4.6
< integer expression >	8.3

8.4.2 General Description

When WRITE is used, the modifier list contains a directive to output a certain marker which later can be searched for using POSITION.

When POSITION is used, 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 sequential files.

8.4.4 Modifier KEY

The modifier KEY is used to specify that a KEY record with a certain identification is to be output or searched for. It has the 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 filled with master spaces up to six characters.

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

NOTE: KEY means the same as KEY (0)

-KEY means the same as -KEY (0)

EXAMPLE:

```
WRITE (FILE('A'),KEY('ABCDEF'))$

WRITE (FILE('A'),KEY('ABCDEFGHK'))$

COMMENT WILL PROCEDURE TWO IDENTICAL KEY RECORDS$
```

EXAMPLE:

```
POSITION (FILE('A'),KEY)$

POSITION (FILE('A'),KEY(0))$

COMMENT HAVE THE SAME MEANING$
```

WRITE outputs a KEY record with its identification given by the parameter on the sequential file.

EXAMPLE:

```
INTEGER I,J,K,L,M$

WRITE(FILE('F'),KEY(I),J,K,LM$
```

EXAMPLE: (cont)

```
COMMENT THE KEY RECORD COMES BEFORE THE DATA RECORD$

REWIND (FILE('F'))$

READ (FILE('F'),I,J,K,L,M)$

COMMENT WILL READ THE VALUES INTO I,J,K,L,M IGNORING THE KEY RECORD$
```

With READ, KEY records are ignored. The action with POSITION is as follows:

- 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 until the KEY with the specified identification is found.
- KEY records are ignored when positioning to EOF or EOI.
- For more information see 8.8.3.

EXAMPLE:

```
BOOLEAN B$

B = TRUE$

POSITION (FILE('B'), 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 POSITION see 8.5.7.

8.4.5 Modifier EOF

The EOF modifier is used to specify that an EOF (end-of-file) record with a certain identification is to be output or searched for. It has the 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 filled with master spaces up to six characters.

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

NOTE: EOF means the same as EOF (0)

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

WRITE outputs an EOF record with its identification given by the parameter on the sequential file. A minus sign has no meaning.

EXAMPLE:

```
ARRAY A(0:500)$

WRITE (FILE('E'),A,EOF('END'))$

COMMENT WILL WRITE OUT THE RECORD CONTAINING THE VALUES OF A AND THEN THE

EOF RECORD WITH IDENTIFICATION WORD 'END'$
```

If the READ operation encounters an EOF record, it will exit via a label in its label list, if such a list exists. See 8.5. The modifier EOF must not be placed in a READ list.

The action with POSITION is as follows:

- 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 (FILE('4'),-EOF)$

READ (FILE('4'),EOFLB,A)$

COMMENT WILL JUMP TO THE STATEMENT WITH THE LABEL EOFLB 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

The EOI modifier is used to specify that an EOI (end-of-information) record is to be output or searched for. It has the form:

EOI or -EOI

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

WRITE outputs an EOI record.

EXAMPLE:

```
COMPLEX ARRAY C(-4:200)$

WRITE (FILE('5')C,EOI)$

COMMENT WILL WRITE ARRAY C TO FILE AND THEN PLACE AN EOI MARKER$
```

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 8.5.

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

8.5 LABEL LIST

The label list allows the user to specify where he would like his program to go 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.

A label list consists of from zero to three labels together or scattered throughout 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, CARDS, or ACARDS

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 for Sequential File Devices

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 for Indexed File Devices

	READ		WRITE	
NUMBER OF LABELS	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 allowed with WRITE	
3	Jump to second label, first label ignored	Jump to third label		

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, CARDS, PRINTER, PUNCH, or Alternate Symbiont Files

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 for Sequential File Devices

NUMBER OF LABELS	ACTION ON END OF SEQUENTIAL FASTRAND OR 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 for Sequential File Devices

POSITION Parameter	Action With POSITION					
Tape Contents	KEY or Arithmetic Expression			EOF		EOI
Number of Labels	EOF	EOI	Physical End of File or Transmission Error	EOI	Physical End of File or Transmission Error	Physical End of File or Transmission Error
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

EXAMPLE:

```
BEGIN
  COMMENT STOP READING DATA CARDS WHEN EOF CARD READ$
  INTEGER ARRAY A(0:1000)$ INTEGER I$
```

EXAMPLE: (cont)

```
LO:  READ (CARDS,A(I),L1,L2,L3)$

     I=I+1$ GO TO LO$

L3:  WRITE ('ERROR IN CARD',I)$  GO TO LO$

L2:  WRITE ('EOF CARD MISSING')$ GO TO STOP$

L1:  WRITE ('ALL CARDS READ')$

STOP:  END$
```

8.6 FORMAT LIST

The format list is a means of specifying how values should be edited. It 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	PARAGRAPH
Implied or free format	8.6.1
Declared format	8.6.2
Inline format	8.6.3

The devices FILE 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

No format is specified before an input/output list. Eighty 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," an area in main storage 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 4.2.4, 4.3, and 4.5.1.

The two exceptions to the preceding rules follow.

- In real constants, a 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 are 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 each free format READ starts with a new image.

EXAMPLE:

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

Data card form is:

```
-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 not all list elements are filled. A transfer function is used here.
X(2,1)	2.0	
X(3,1)	3.0	
X(4,1)	4.0	
X(5,1)	5.0	All characters after an * are ignored.
X(1,2)	6.0	
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	
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

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 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:

```
BEGIN
  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 Declared Format

A specific sequence of phrases is declared and an identifier attached, which can be used in the format list. It has the 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)$
```

8.6.3 Inline Format

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

EXAMPLE:

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

8.6.4 Format Phrases with WRITE

Format phrases are used with WRITE, as shown in Table 8-1, to specify the output form of each parameter as well as the exact position for the placement of the value of the parameter.

A format phrase has the form:

Qw.d

or $Q(E_1,E_2)$

- Q represents one of the letters given below. Qw.0 may be abbreviated to Qw, and Q0.0 or Q0 to Q.
- E_1 must be an arithmetic expression with the same restrictions as w.
- E_2 must be an arithmetic expression with the same restrictions as d.
- w and d are positive integers and are defined in Table 8-1.

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.

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 are:

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.

The action when the end of the print buffer is reached is:

1. 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.
2. For device CARDS or PUNCH, the limit is column 80.

Table 8-1. Format Phrases for WRITE

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Activate	Device implied or PRINTER								
Aw.d or A(E1,E2)	Print 1 line	Skip w lines before printing	0	63	Skip d lines after printing	0	31		Nonediting does not require a parameter
	Device CARDS or PUNCH								
	Punch 1 card	ignored			ignored				
	Device CORE								
	Transfer as many characters from the print buffer into the string as the length of the string or print buffer allows	ignored			ignored				
Boolean	Devices implied, PRINTER, CARDS, CORE, PUNCH								
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 characters used in the print buffer)	1	132 80 for CARDS	Not allowed			Left-justified	BOOLEAN

Table 8-1. Format Phrases for WRITE (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Decimal	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Dw.d or D(E1,E2)	Transfers a decimal number with d digits after the decimal point - leading zeroes suppressed, minus sign if negative.	Field width	2	63	Provide d digits after decimal point	0	31	Right-justified	INTEGER REAL REAL2 COMPLEX
Eject	Devices implied, PRINTER								
Ew or E(E1)	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.) Devices CARDS, CORE, PUNCH Ignored	Logical line number on page	1	72	Not allowed				Nonediting does not require a parameter
Free	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Fw or F(E1)	Transfer a field of w characters in free format. See 8.6.1.	Field width	1	132 (80 for CARDS or PUNCH)					INTEGER REAL BOOLEAN COMPLEX REAL2 STRING

Table 8-1. Format Phrases for WRITE (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Integer	Device implied, PRINTER, CARDS, CORE, PUNCH								
Iw.d or I(E1,E2)	Transfer 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)
Absolute position to column	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Jw or J(E1)	The next phrase will start from column w.	Column number	1	132 80 for CARDS PUNCH	Not allowed				Non-editing
Middle string	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Mw or M(E1)	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 rightmost L-w characters of the parameter are lost.	Field width	1	132 80 for CARDS PUNCH	Not allowed			Center-justified	STRING

Table 8-1. Format Phrases for WRITE (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Left-justified Integer	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Nw or N(E1)	Edit an integer decimal number left justified with a leading minus sign if negative or a leading space if positive	Field width	1	63	Not allowed	0	10	Left-justified	Same as 1 phrase
Real	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Rw.d or R(E1,E2)	Edits the parameter into the form ±X.XXX···X,±XX d significant digits Note: w > d+6. If the parameter is REAL2 and needs three digits for the exponent, the no. of significant digits will be d-1.	Field width	7	63	Number of significant digits	1	31	Right-justified	INTEGER REAL REAL2 COMPLEX
String	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Sw or S(E1)	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

Table 8-1. Format Phrases for WRITE (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Real zero gives blanks	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Uw.d or U(E1,E2)	If value of the parameter is exactly zero then treat as Xw, otherwise treat as Dw.d	Field width	1	63	Ignored				
		Field width	1	63	Provided digits after the decimal point	0	31	Right-justified	INTEGER REAL REAL2 COMPLEX
Integer zero gives blanks	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Vw.d or V(E1,E2)	If value of the parameter is exactly zero then treat as Xw, otherwise treat as Iw.d	Field width	1	63	Ignored				
		Field width	1	63	Base for integer	0	10	Right-justified	INTEGER REAL REAL2 COMPLEX BOOLEAN
Place blanks	Devices implied, PRINTER, CARDS, CORE, PUNCH								
Xw or X(E1)	Place w blanks into the print buffer	Number of blanks	1	132 80 for CARDS	Not allowed				Non-editing

Table 8-1. Format Phrases for WRITE (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
String Constant String of characters enclosed in ' '	Devices implied, PRINTER, CARDS, CORE, PUNCH Place the characters in the number of columns required.								Non-editing

EXAMPLE:

Differences between D, R and U phrases

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

Print lines

```
      31.4159     3.1416,+01     31.4159
           0              0
```

EXAMPLE:

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

```
      -31415-31415            31415
           0 0
```

EXAMPLE:

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

Format phrases are used to inform READ, as shown in Table 8-2, 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.

A format phrase has the form:

Qw.d

or

Q(E1,E2)

- Q represents a formatting character (see following explanation).
- E1 must be an arithmetic expression with the same restrictions as w.
- E2 must be an arithmetic expression with the same restrictions as d.
- w and d are positive integers and are defined in Table 8-2.

The following 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.

Table 8-2. Format Phrases for READ

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Activate	Devices implied, CARDS								
A	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
	Device CORE								
	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
Boolean	Devices implied, CARDS, CORE								
Bw or B(E1)	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

Table 8-2. Format Phrases for READ (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Decimal	Devices implied, CARDS, CORE								
Dw.d or D(E1,E2)	Accept a numeric constant in the form of INTEGER, REAL or REAL2 as described in 4.2.2. Make it negative if preceded by a minus sign. 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 insert a decimal point to the right of the (d+1) at digit (counting from the right) in the field, else ignore	0	31		INTEGER REAL REAL2 COMPLEX
Eject	Ignored by all devices								
Ew or E(E1)									
Free	Devices implied, CARDS,CORE								
Fw or F(E1)	Read the next w columns in the manner described in 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

Table 8-2. Format Phrases for READ (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Integer	Devices implied, CARDS, CORE								
Iw.d or I(E1,E2)	(1) Accept a numeric constant in the form of INTEGER, REAL, or REAL2 as described in 4.2.2. Make it negative if preceded by a minus sign. (2) Give the value a type according to the form of the number read (see 4.2.3 for form of numbers). (3) Convert the number to integer. (4) Convert the result to the type of the parameter.	Field width	1	63	Base of integer (d=0 is the same as d=10)	0	10		INTEGER REAL REAL2
Position to column	Devices implied, CARDS, CORE								
Jw or J(E1)	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 phrases
Middle String									
Mw or M(E1)	Exactly the same as S.								

Table 8-2. Format Phrases for READ (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
Integer									
Nw or N(E1)	Exactly the same as Iw.								
Real									
Rw.d or R(E1,E2)	Exactly the same as D.								
String	Devices implied, CARDS, CORE								
Sw or S(E1)	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

Table 8-2. Format Phrases for READ (cont)

PHRASE	ACTION	w or E1			d or E2			POSITION IN FIELD	ALLOWED TYPES OF PARAMETERS
		MEANING	MIN	MAX	MEANING	MIN	MAX		
No change if blanks real Uw.d or U(E1,E2)	Devices implied, CARDS, CORE (1) If the field reserved is completely blank treat as Xw. (2) Otherwise treat as Dw.d.	Field width	1	63	Ignored				INTEGER REAL REAL2 COMPLEX
		Field width	1	63	Same as for D	0	31		
No change if blanks integer Vw.d or V(E1,E2)	Devices implied, CARDS, CORE (1) If the field reserved is completely blank treat as Xw. (2) Otherwise treat as Iw.	Field width	1	63	Ignored				INTEGER REAL REAL2 COMPLEX
		Field width	1	63	Same as for I	0	10		
Blanks Xw or (E1)	Devices implied, CARDS, CORE Skip the next field of w columns.	Field width	1	80	Not allowed				Non-editing
String Constant String of characters enclosed by ' '	Completely ignored.								Non-editing

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 occurred and the line number of the program being executed.

Common errors are:

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.)

8.6.6 Repeat Phrases

8.6.6.1 Definite Repeats

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. It has the form:

```
nQw.d

n(Qw.d,Qw.d,.......Qw.d)

:E:(Qw.d)

:E:(nQw.d,:E:(Qw.d),:E:(nQw.d))

etc.
```

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

The following rules apply to definite repeats:

1. The expression E is evaluated when the repeat phrase is activated. That is when the format phrase is required, before the parameter is evaluated.
2. If E > 0 the format phrase (S) are repeated that many times. If E = TRUE the phrases are taken once.
3. If E ≤ 0 or E = FALSE the format phrase(s) which this repeat controls will be skipped.

EXAMPLES:

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

8.6.6.2 Indefinite Repeats

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.

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.

NOTES:
1. 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.
2. 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.
3. Attempts to cause an indefinite repeat of a format containing only nonediting 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))$
```

EXAMPLES: (cont)

```
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

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.

The list may have two forms:

Declared list

Inline list

8.7.1 Inline List

The inline list gives the input or output statement a list of expressions to or from which values may be transferred. 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, ARE ALL MEMBERS OF INLINE LISTS$
```

8.7.2 Declared List

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

8.7.3.1 Arrays

An array identifier may be used without subscripts in a list, i.e., every element in the array is to be used in the list.

For multidimensional arrays, the leftmost subscript varies most frequently, i.e., a two-dimensional array will be decomposed by columns.

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),
```

EXAMPLES: (cont)

```
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)$
```

8.7.3.2 Other Expressions

The expression is evaluated at the time the list element is referenced. Expressions other than variables or array names may not be used as list elements in an input list.

8.7.3.3 Format in Lists

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

8.7.3.4 List with MAX and MIN

The parameters to MAX and MIN are given in the form of declared or inline lists, see 7.4.1.

8.7.4 Sublists

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

8.8.1 READ

READ is used to specify that values are to be input according to the given parameters. It has the form:

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

All devices are allowed except PRINTER, APRINTER, PUNCH, and APUNCH (see 8.3). Up to three labels may be used. See 8.5.

8.8.2 WRITE

WRITE is used to specify that values are to be output according to the given parameters. It has the form:

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

All devices are allowed, see 8.3.

EXAMPLE:

```
WRITE(FILE('A'),ERRLB,EOF('XYZ'),X,Y,Z)$

WRITE(CORE(S), <<3R12.2,A>> ,X,Y,Z)$
```

Only one label is allowed. See 8.5.

8.8.3 POSITION

POSITION is used as follows:

- Sequential file - To position a file to a previously written KEY or EOF record, to the end of information or advance it over a given number of records.
- Indexed file - To position a file to a position specified by a file index.

The POSITION statement has the form:

```
POSITION(FILE( < filename > ),< modifier list>,<integer expression>,<label list>)$
POSITION(FILE(<filename>, < fileindex>))$
```

Only FILE or DRUM are allowed as devices. See 8.3. Up to three labels may be used. See 8.4 and 8.5.

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. Certain tape units cannot be positioned backwards. See 8.4.3.

8.8.4 REWIND and REWINT

REWIND positions a file to its starting position, and releases its buffers. REWINT does the same as REWIND except when the file is on magnetic tape. In that case, the tape is also locked, so that it can no longer be used.

```
REWIND(FILE( < filename > ))$

REWINT(FILE(< filename > ))$

REWIND(FILE( < filename > ,< fileindex > ))$
```

Only FILE or DRUM devices are allowed with these operations. See 8.3.6.

8.8.5 MARGIN

MARGIN is used 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. It has the form:

MARGIN(<control string>)

- control string is a string containing one or more control functions.

Spaces are ignored prior to the first, or between functions. Each function begins with a single letter, followed by a comma, followed by any special information required, and terminated by a period. The format of the information character string varies according to the function but must not contain a period.

The following control functions are allowed:

- L - Space printer to logical line nn, where logical line is defined as the line number relative to the top margin setting (see M following). All line positioning and printing is performed within the defined margin settings. (The bottom logical line of a page is identical to the top logical line -1 of the next page.) Positioning to a logical line on printers with space-print operation is to logical line n - 1; therefore when n = 1, the logical line setting is the last line of the current page. This is also true when n = 0, or when n is greater than the length of the logical page. When n is less than or equal to the current line of the current page, the succeeding page is positioned to the logical line n - 1. The format of this function is:

  L,nn.

- H - Initiate heading printing. This function provides the user with an automatic means of printing a heading on each succeeding page of his print file. The format of this function is:

  H, option, page , text of heading.

  If the option field contains the letter X, a page and date will not be printed as part of the heading. Option N turns the heading off. A page count is maintained by the processing symbiont. When the page field is blank, the page count current to the field is used to begin page numbering. When coded, page is made the page number. In addition to the page number, the current date is included in the heading, and both will appear in the upper right corner of each page. This position of the heading is the second line above logical line 1. If the upper margin is one line or non-existent, no heading is printed. As many as 17 words of heading text may be supplied.

- M - Set margins. This function supplies the information for readjusting page length and top and bottom margins. The standard print page definition is 66 lines per page with a top margin setting of six lines, and a bottom margin setting of three lines. Note that the top and bottom margins refer

to the number of blank lines at the top and bottom of the page respectively. Thus the standard margin setting is 66,6,3. giving 57 printable lines. The page definition is assumed at the beginning of each print file. When the M function is used, a page alignment procedure is initiated with the page length parameter. This function is also used to return to the standard page length. The format of this function is:

M, length, top, bottom.

- W - Set maximum line width to allow error checking on image length at run-time. The standard of 22 words is assumed unless the W control is used. The format of the function is:

W, width.

where width specifies the maximum line width in words. The program is errored out when W is exceeded.

- S - Special form request. This function enables the user to instruct the operator to load a special form required to process the print of punch file. The format of this function is:

S, message text.

where the message text can be up to ten words long. When this function is encountered by the processing symbiont, the message is displayed on the operator's console in the form:

run ID/filename c/u options

message text

The user's message text is displayed on the line following the symbiont message. The options available to the operator for answering the message depend on the symbiont. The following options are included in the 0755 HSP, Card Punch and the 1004 Printer and Card Punch symbionts:

A - Begin processing the output file.

Q - Return file to symbiont queue. The print or punch file will be passed temporarily and placed behind the next file of this symbiont queue.

EXAMPLE:

Set the margin to print 72 lines per page:

```
MARGIN('M,72,0,0.');
```

9. COMMENTS, UTILITY STATEMENTS, AND OPTIONS

9.1 COMMENTS

The use of explanatory messages is encouraged to aid readability of the program and to help in finding errors in the source text. The following are methods of using comments:

- After BEGIN or any $ or ; the following construction may be placed.
- COMMENT any characters not including ; or $ followed by ; or $
- After END comments can be placed. However, the characters ; or $ or the words END or ELSE cause the ending of the comment.
- In a procedure declaration, comments may be placed in the formal parameter list by substituting the comma with 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
```

EXAMPLE: (cont)

```
SHOW (I,50+I)$

END OF THIS PROGRAM SHOWING COMMENTS$
```

NOTE: A comment may come before the first BEGIN of a program

9.2 UTILITY STATEMENTS

Utility statements are used to produce actions which have no effect on the results of the computations of the program. These actions may occur at compiletime or runtime. Some of these actions are also provided by the use of option letters, either on the processor card or on the execute card. In this paragraph only actions being explicitly performed by the NU ALGOL compiler or runtime system will be described. Options for the processor interface routines and the collector are therefore omitted. One of the most important uses of utility statements is for debugging purposes.

Utility statements have the following form:

SET<list of directives separated by commas>;

RESET<list of directives separated by commas>;

There is one directive for each action. Actions are turned on by the SET statement and off by the RESET statement. Utility statements may occur anywhere in a NU ALGOL program where a COMMENT may be placed. Depending upon the contained directives, the SET statement may be required to occur only before the first BEGIN of the program.

Some directives have only meaning in a SET statement and some may have parameters supplied to give better control of the actions.

Directives with names containing the word TRACE will dynamically produce output during execution. Except for TIMETRACE this output will be intermixed with output produced by, for instance, WRITE statements. Directive names containing CHART or TABLE imply some kind of accumulated or condensed information being printed at program termination.

The directives are listed below. Corresponding option letters are given where applicable. RESET is permitted unless otherwise indicated.

1. ACCEPTERRORS — Before first BEGIN; Option A; RESET not meaningful

   The compiled relocatable element will not be marked in error even if compile-time errors are found. Compiletime warning messages are suppressed.

2. ASSEMBLERLISTING Option L

List the object code produced by the compiler together with the source text. The code produced will appear just before the corresponding source line. If NOSYMBOLIC also is in effect, the source text will not be listed.

3. BLOCKMESSAGE Option B

List during compilation the serial number and level number at the beginning and end of each block in the program.

4. COMPILEABORT Before first BEGIN
Option X
RESET not meaningful

Abort the entire run when the compilation is finished if compiletime errors are found.

5. ERRORDUMP

When a runtime error occurs an edited dump with symbolic identifier names is made of all variables which are currently accessible. The directive should be placed immediately before the BEGIN of the first block of which a dump is wanted.

EXAMPLE:

```
SET ERRORDUMP;

BEGIN

  INTEGER I,J;

  REAL A;

  READ(I,J);

  A=ABS(I*J);

  RESET ERRORDUMP;

  BEGIN

    REAL B;

    B=SQRT(-A);

  END;

END;
```

The statement B=SQRT(-A) produces the error message "ARGUMENT OUT OF RANGE". Then a dump is made of the variables of the outermost block, i.e., I, J and A. B is not dumped because ERRORDUMP was turned off for the innermost block.

6. EXTERNALCOMPILATION

Before first BEGIN
Option E
RESET not meaningful

The names of all procedures declared in the outermost block are placed in the table of external definitions. The directive is required for separately compiled NU ALGOL procedure elements. See also 7.3.2.

7. GOTOTRACE

A message is printed each time a GOTO statement is executed and each time a label is entered within the active range of the directive. The message contains line numbers of the GOTO statement and the label declaration and the name of the label entered.

8. IFTABLE

A summary of the execution of IF statements and expressions is printed at program termination. The table shows how many times each IF clause has been executed and how many times the Boolean expression has been TRUE and how many times FALSE.

Each IF clause is identified by a line number and an IF serialnumber within the line.

9. IFTRACE

A message is printed each time an IF clause is executed within the active range of the directive. The message identifies the IF clause by a line number and an IF serial number within the line and gives the value of the Boolean expression.

10. LINETRACE

The source line number is printed for each statement executed. This listing shows the sequence in which the statements are executed.

Only source lines producing code will be traced. Comments and declarations of simple variables will not be listed.

11. LISTINPUT Option I on execute card

List data images as they are read during execution. The line number of the executing READ statement and six masterspaces are inserted in front of the image.

12. NOLINENUMBERS Option Z

The compiler suppresses generation of line information. The program will the. not keep track of the line numbers being executed so that they will be missing from runtime error messages. The use of this directive saves time and main storage space (about one instruction per source line). It should be used in production programs only.

13. NOSUBSCRIPTCHECK (<list of array identifiers>) Option O or R

Subscriptchecking will be suppressed in the range where the directive is active. Only the listed arrays will be affected. If no parameters are supplied, all arrays referenced in the range will lack subscript checks.

Removing the subscriptcheck may save 20 to 30 percent of the execution time for programs doing mostly array work. The feature should not be used while debugging.

EXAMPLE:

```
BEGIN

  INTEGER I,J;

  ARRAY A,B,C(1:100);

  SET NOSUBSCRIPTCHECK(A);

  READ(J,FOR I=(1,5,100) DO(A(I),B(A(I))));

  COMMENT SUBSCRIPTS FOR B ARE CHECKED;

  RESET NOSUBSCRIPTCHECK(A);

  FOR I=(2,1,J) DO C(2*I)=A(I)+B(I//2);

  COMMENT ALL SUBSCRIPTS ARE CHECKED;

  SET NOSUBSCRIPTCHECK;

  FOR I=(1,1,100) DO A(I)=B(I)-C(I);

  COMMENT NO SUBSCRIPTS ARE CHECKED;

END;
```

14. NOSYMBOLIC Option N

Suppress listing of source text. If neither SYMBOLIC nor NOSYMBOLIC appear in the program, the latter is taken as the default.

15. NOWARNINGS Option V

Suppress compiletime warning messages.

16. PROCEDURETRACE (<list of procedure identifiers>)

The procedures of the list are traced by printing a message for each occurrence of a procedure entrance or exit. Only NU ALGOL procedures can be traced. The messages identify the place of the procedure call. The exit message also contains the amount of time spent within the procedure. The time is total time including time spent in procedures called within the procedure.

When there is no parameter list supplied to the directive, all procedures called within the active range of the directive will be traced.

17. PUNCHASSEMBLER Option F

The symbolic assembler code produced by the compiler is listed and output to the punchfile. This file can then be processed by the assembler.

18. SYMBOLIC Option S

List the source input text with line numbers and numbering of all BEGIN and END statements.

19. TIMECHART (<integer expression>)

A bar chart of asterisks showing the total amount of execution time spent on each source line is printed at program termination. At the top of the chart the total execution time for the active range of the directive is printed. Each bar is preceded by a source line number and a percentage showing how much of the total time was spent on the line.

TIMECHART is implemented by sampling the line counter at regular intervals. The parameter <integer expression> supplies the sampling interval in milliseconds of CPU time. It must be larger than two milliseconds. If no parameter is given, two milliseconds is assumed.

Each asterisk in the chart corresponds to one sampling. If too many samplings are done, the internal buffer will overflow. This produces an error message and the sampling stops. At an interval of two milliseconds the buffer will overflow in a few seconds. For larger execution times the sampling interval must be increased so as to produce fewer samplings.

Because of the sampling technique used, the chart will not be completely accurate and it will vary somewhat from one execution to another of the same program. The accuracy increases with the number of samplings.

20. TIMETRACE (<integer expression>)

The directive will give a bar chart after the program has terminated. Each bar will be marked with the associated line number in the NU ALGOL program and will show how much time was spent on that line of source text each time it was executed. TIMETRACE can be set for the whole program or only parts of it.

If the <integer expression> is present it will specify a threshold value (in milliseconds). All lines with execution times less than this threshold will not appear in the chart.

It is important to be aware of the inaccuracies inherent in the measurements. For instance, on the UNIVAC 1108 Processor, the clock is incremented every 0.2 milliseconds which is then the smallest measurable time unit. Only source lines with execution times in the order of one millisecond or larger can therefore be meaningfully traced. This limits the usefulness of TIMETRACE in most cases to the timing of procedure calls.

21. TIMING

Before first BEGIN
Option T
RESET not meaningful

When compilation is finished, compilation times are given for each of the four passes of the compiler and for the total. The number of words used on the intermediate file for output between passes, is also printed.

22. VALUEDUMP (<integer expression>)

This directive is similar to ERRORDUMP, but edited dumps will be produced each time the directive is encountered. After the dump is done, the execution continues normally. The <integer expression> specifies the number of nested active blocks to be dumped, counting from the current block and outwards. If no parameter is supplied or the parameter is larger than the number of active blocks, all active blocks will be dumped.

23. VALUETRACE (<list of simple or array identifier>)

This directive activates a dump of a variable each time it occurs on the left-hand side of an assignment or in a READ statement. The format of the dump is:

xx<line number of statement>:<identifier>=<value>

String values will be enclosed in quotes. The parameter list specifies which variables to trace within the range of the directive. If no parameter list is supplied, all variables will be traced.

EXAMPLE:

```
BEGIN
  INTEGER M,N;
  SET VALUETRACE;
    READ(M,N); COMMENT BOTH M AND N ARE DUMPED;
  RESET VALUETRACE;
  BEGIN
    ARRAY A,B(1:100);
    REAL X,Y;
    SET VALUETRACE(A,Y);
    READ(A); COMMENT DUMP A;
    FOR N=(1,1,100) DO
    BEGIN
      X=Y=A(1)+A(N); COMMENT DUMP Y ONLY;
      RESET VALUETRACE(Y);
      SET VALUETRACE(B);
      Y=B(N)=A(N)**2; COMMENT DUMP B(N) ONLY;
    END;
```

24. XQTABORT — Option X on execute card

Abort the rest of the run if errors occur during execution.

9.3 OPTIONS

It is possible to control certain actions of the ALGOL compiler and run-time system by placing a specific option letter after the masterspace on the ALGOL processor card or the XQT card. Only options that are particular to NU ALGOL are described in this section. For the use of other options, consult the manual of the relevant operating system. At compile time, 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 statement.

9.3.1 Processor Card Options

The following are the available options on the processor card:

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

B - List serial number and level number at the beginning and end of each block in the program during compilation.

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

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

L - The assembler code produced by the compiler will be listed. If there is no N option, the source code will also be listed. The instructions resulting from each line of ALGOL text will appear just before the corresponding source line.

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 whether the subscript being used is within the bounds declared for the array. It is suggested that this option should not be used during debugging. Production programs can benefit greatly from the saving in time when the check is removed.

S - The ALGOL source statements are listed.

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.

Y - Suppress the warning, "THIS VARIABLE HAS NOT BEEN ASSIGNED A VALUE"

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

9.3.2 XQT Card Options

The following are the available options on the XQT card:

A - Accept the program for execution even though errors have been found during compilation or allocation. If compile-time errors have occurred, 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

I - List data images as they are read. The line number of the READ statement and are inserted in front of the image.

X - Abort the rest of the run if errors occur.

10. ERROR MESSAGES

10.1 GENERAL

The compiler tries to find 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, all other errors diagnosed should be corrected. In most cases, the internal error will then disappear.

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.

There are three levels of errors.

1. 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

   Warnings can be suppressed by using the V option or as a side effect of the A or N options.

2. 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.

3. 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.

10.2 COMPILE-TIME ERROR MESSAGES

Compile-time error messages are given in Table 10-1.

Table 10-1. Compile-Time Error Messages

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
1	Illegal number	The number does not conform to the syntax of 4.2.3.
2	Illegal character	Some special characters cannot be used outside strings or comments. (See 2.1.)
3	Correction card error	Line number on correction cards is not in ascending order.
4	Improper use of reserved identifier	Reserved identifiers (see 2.2) may only be used with their special meaning.
5	Too long string	String constants may not have more than 200 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 5.5.)
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 7.1.4.)
12	Pass 1 stack overflow	An internal compiler error usually caused by other errors or a too large program.
15	Double specification of <name of variable>	A parameter to a procedure has been specified twice. (See 7.1.)
16	Illegal value specification of < name of variable >	LABEL, LIST, FORMAT, SWITCH and PROCEDURE cannot be given a value specification.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
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 inserted	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.
26	Interphase 1 error	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.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
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 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.)
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 in 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 subscript bounds in array declarations.
40	*Warning* Do you want to compare constants?	Possible punching error.
41	Improper type before THEN	Only Boolean expressions are allowed before the delimiter THEN.
42	Improper relation between complex expressions	Complex numbers can only be compared for equality or nonequality.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
43	Undefined transfer function	An implicit nonexistent conversion has been called for. (See 7.3.)
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.
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 nonintegers 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	The number will depend upon how much the compiler is allowed to expand dynamically. Normally, several thousand identifiers are allowed for.
54	Pass 2 stack overflow	Internal compiler error. Check for other errors which may have caused the compiler confusion. The program may have too many declarations.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
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 than the scratch area on drum.
57	Improper format construction	Some rule for formats has been broken (see 8.6).
58	Zero replicator	Although replicator expressions may have the value zero, the constant replicator zero has no meaning.
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 8.6.)
62	*Warning* Missing activate within indefinite repeat	Indefinite repeat formats usually require an A-phrase to perform properly.
63	*Warning* 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	*Warning* Time consuming conversion to integer subscript in array <array name>	It is allowable to use noninteger expressions for subscripts, but it is very slow.
66	Illegal format character	Only certain characters are meaningful within a format. (See 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 source input file. Usually a hardware error.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
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	Operand stack underflow	Internal compiler error - check for other possible errors.
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} or 10^{308} respectively, 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 4.5.)
76	Operator stack underflow	Internal compiler error - check for other possible errors which could have confused the compiler.
77	*Warning* 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 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.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
80	*Warning* 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.
81	Division by zero	Division by zero has been attempted in a constant expression being evaluated 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 7.3.4.2.
86	Prototype table overflow	The program contains too 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 strings with different bounds.
90	*Warning* Parameter to NOSUBSCRIPTCHECK is not ARRAY <identifier name> at line <line of declaration>	The parameter is a label, array switch, format, list or procedure.
91	*Warning* Parameter to VALUETRACE cannot be assigned a value <identifier name> at line <line of declaration>	The parameter is a label, array switch, format, list or procedure.
92	*Warning* Parameter to PROCEDURETRACE is not a procedure <identifier name> at line <line of declaration>	The parameter is a label, array switch, format, list or procedure.

Table 10-1. Compile-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
93	*Warning* This variable has not been assigned a value: <identifier name> at line <line of declaration>	The compiler has not detected that a value was assigned to this variable.
94	Too many blocks and/or procedures; only 60 allowed.	More than 60 blocks and/or procedures.
95	A maximum of 64 nested IF statements is allowed.	More than 64 nested IF statements.

10.3 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 occurred. Table 10-2 lists the run-time error messages.

Table 10-2. Run-Time Error Messages

ERROR 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 parameter. 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 allowable has been produced.
7	Attempted division by zero	An attempt was made to divide by zero.
8	(Not Used)	
9	Illegal operation	Missing external procedure or incorrect return from a FORTRAN or assembly language procedure.
10	Result undefined for conversion	The result produced by a transfer function is not a meaningful value.
11	(Not Used)	

Table 10-2. Run-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
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. Too many files requiring buffer space may be open at one time.
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 or an operand in an exponentiation is not within the limits accepted.
16	Subscript out of range	The subscript computed for an array element does not fall within the bounds specified in the array declaration.
17	(Not Used)	
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 produced a lower bound that is greater than the upper bound.
20	(Not Used)	
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 incorrectly. 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.

Table 10-2. Run-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
22	Improper parameter	Improper parameter in size or sign.
23	(Not Used)	
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 proceed as far as the first error.
26	Improper type of controlled variable	The controlled variable of a FOR statement is a formal parameter which is not VALUE specified and 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 sequential file record is not correct. Possible hardware error or incompatible file format.
30	File error	I/O attempted beyond file limits, or a transmission error has occurred.
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 devices FILE or TAPE is longer than the record input from the file.
34	Formats are not allowed with FILE, TAPE or DRUM	Devices FILE, 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.

Table 10-2. Run-Time Error Messages (cont)

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
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.
40	Sequential file referenced as indexed or indexed file referenced as sequential	The same file name cannot appear with both one and two parameters to FILE.

APPENDIX A. BASIC SYMBOLS

Out of the 64-character set of the UNIVAC 1100 Series processors, 55 characters are recognized by the NU ALGOL compiler as being meaningful within an ALGOL program. (See 2.1.) The remaining 9 characters have no inherent meaning and are allowed only within strings and comments. 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). Table A-1 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 symbol is shown.

Table A-1. 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 language are represented by means of the preceding characters. Table A-2 shows these symbols along with the corresponding symbols of the ALGOL 60 reference language.

Table A-2. NU ALGOL Basic Symbols

ALGOL 60	NU ALGOL	ALGOL 60	NU ALGOL
true	TRUE	step	STEP
false	FALSE	until	UNTIL
+	+	while	WHILE
-	-	comment	COMMENT
x	*		SET
/	/		RESET
÷	//	(	(
↑	**	)	)
<	LSS	[	(or [
≤	LEQ	]	) or]
=	EQL		<
≥	GEQ		>
>	GTR		<<
≠	NEQ		>>
≡	EQIV	'	' ("in strings)
⊃	IMPL	,	' ("in strings)
∨	OR	begin	BEGIN
	XOR	end	END
∧	AND	own	
¬	NOT	Boolean	BOOLEAN
go to	GO TO	integer	INTEGER
	or GOTO or GO	real	REAL
if	IF		REAL2
then	THEN		COMPLEX
else	ELSE		STRING
for	FOR	array	ARRAY
do	DO	switch	SWITCH
	OPTION		FORMAT
	OFF		LIST
'	'		LOCAL
.	.		EXTERNAL
10	& or &&		ALGOL
:	: cr ..		FORTRAN
;	; or $		LIBRARY
:=	= or :=		SLEUTH
		procedure	PROCEDURE
		label	LABEL
		value	VALUE

APPENDIX B. EXAMPLES OF PROGRAMS

This appendix contains some simple examples illustrating the use of UNIVAC 1100 Series NU ALGOL. Each example has been run, 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) $
      WRITE ('COEFFICIENTS',B)$ WRITE ('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
```

```
BEGIN
   COMMENT                    EXAMPLE 5

   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
```

APPENDIX C. JENSEN'S DEVICE AND INDIRECT RECURSIVITY

The purpose of this appendix is to acquaint the reader with two interesting progr ming 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 Annual Review in Automatic Programming, Pergamon Press, London, 1963.

Jensen's Device comprises the use of two parameters in a procedure call, in which one parameter 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. UNIVAC 1106/1108 ALGOL AND NU ALGOL DIFFERENCES

D.1 DIFFERENCES AND RESTRICTIONS

This appendix lists the differences between the UNIVAC 1106/1108 ALGOL and the NU ALGOL program languages.

D.1.1 External Procedures

- External procedures compiled using the UNIVAC 1106/1108 ALGOL compiler cannot be run together with ALGOL programs compiled using the NU ALGOL compiler (and vice versa).
- External procedures compiled using the NU ALGOL compiler must have an E-option on the compiler control card (ALG card).
- An external procedure must be terminated with a ; or $ as must any other declaration.
- The declaration EXTERNAL NON-RECURSIVE PROCEDURE is not allowed.
- The declarations for external procedures coded in assembler language are EXTERNAL ASSEMBLER PROCEDURE or EXTERNAL LIBRARY PROCEDURE depending on the type of parameter transmission.

D.1.2 Declarations

- The declaration OWN is not allowed.
- The declaration OTHERWISE is not allowed.
- Two new reserved words have been introduced: OPTION and OFF.
- A procedure may have at most 63 parameters.

D.1.3 Formats

- In input or output statements, the format identifier must come before the list to which it applies.
- The format phrase T is not allowed.

D.1.4 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.

(3) Standard procedures are not recursive.

D.1.5 FOR Statements

(1) The controlled variable may only be of type REAL or INTEGER.

(2) If the controlled variable is a subscripted variable, the initial value of the subscript expression (before entering the loop) determines which member of the array becomes the controlled variable.

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

D.1.6 IF Statements

- An IF expression used in an arithmetic expression must be enclosed in parentheses.

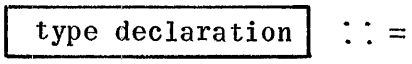
D.1.7 Miscellaneous

- All programs with the exception of external procedures must be enclosed with BEGIN END$.

- In a multiple assignment statement, all of the variables to which the assignment is being made must be of same type.
- The value specification must be placed in front of the type specifications.
- Use of the device DRUM is somewhat different. See 8.3.6.2.
- In input and output, tapes 21 through 27 and CREAD and RREAD are no longer implemented. Continuous reading and re-reading may be done as shown in 8.3.4.
- The statement REWINT (FILE(1)$ must be used instead of REWIND (FILE(),INTERLOCK)$.
- When errors or EOF-conditions are detected during I/O and no labels are provided, the program is terminated with an appropriate message.
- Positioning to a KEY is halted if an EOF is encountered. See Section 8.5.7.
- When a string is a parameter to a FORTRAN procedure, the address of the string itself is transmitted, not the address of the string descriptor.
- Numbers on data cards may not contain spaces.
- Strings in free format WRITE are not printed on separate lines.
- In a Boolean expression, only those operands necessary for determining the result are evaluated.
- If an integer number is input with Dw-d or Rw-d formats, a decimal point is inserted to the right of the (d+1)st digit (counting from the right) in the field.

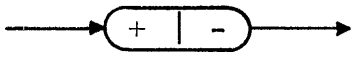
APPENDIX E. SYNTAX CHART

This appendix summarizes the syntax of NU ALGOL in chart form. The use of the chart portion of the appendix 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,

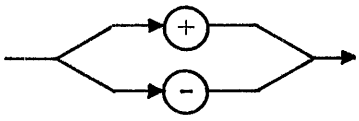


The definition consists of a series of boxes connected by lines indicating the flow of symbols which define the concept. Two kinds of boxes are distinguished: those with round corners (or circles) and those with square corners. The round cornered boxes contain symbols that stand for themselves. Square 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. The 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
$	;

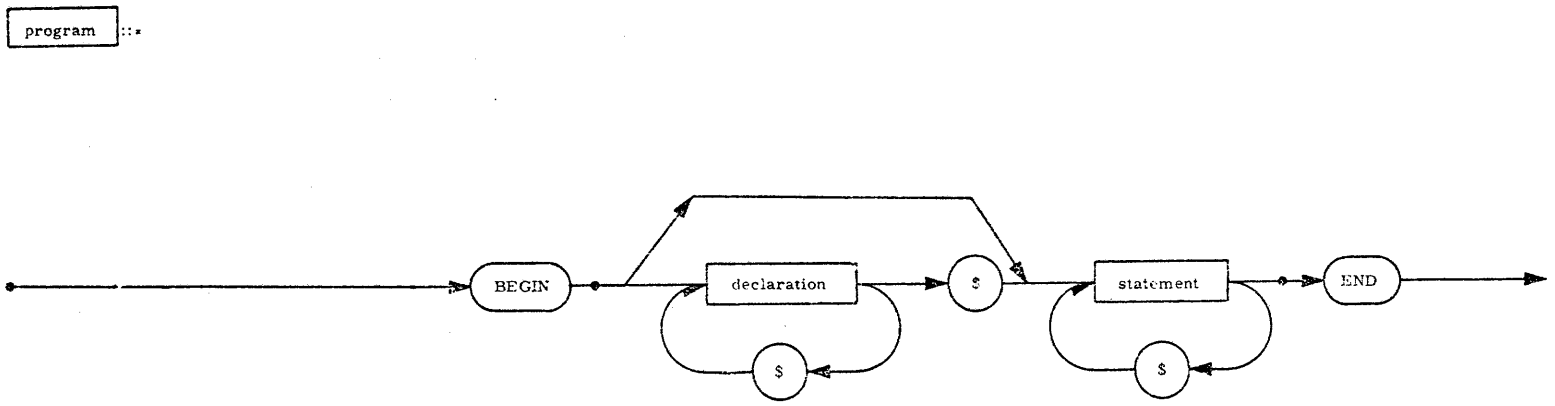
In addition, comments may be inserted in the program by means of the following equivalences:

- $ COMMENT <any sequence not containing a $> $ equivalent to $

- BEGIN COMMENT <any sequence not containing a $> $ equivalent to BEGIN
- END < any sequence not containing END or ELSE or $> equivalent to END

This chart makes no mention of the use of spaces within ALGOL. A space has no meaning in the language (outside of strings) except that it must not appear within numbers, identifiers, or basic symbols, and must be used to separate contiguous symbols composed of letters or digits. Spaces may be used freely to facilitate reading.

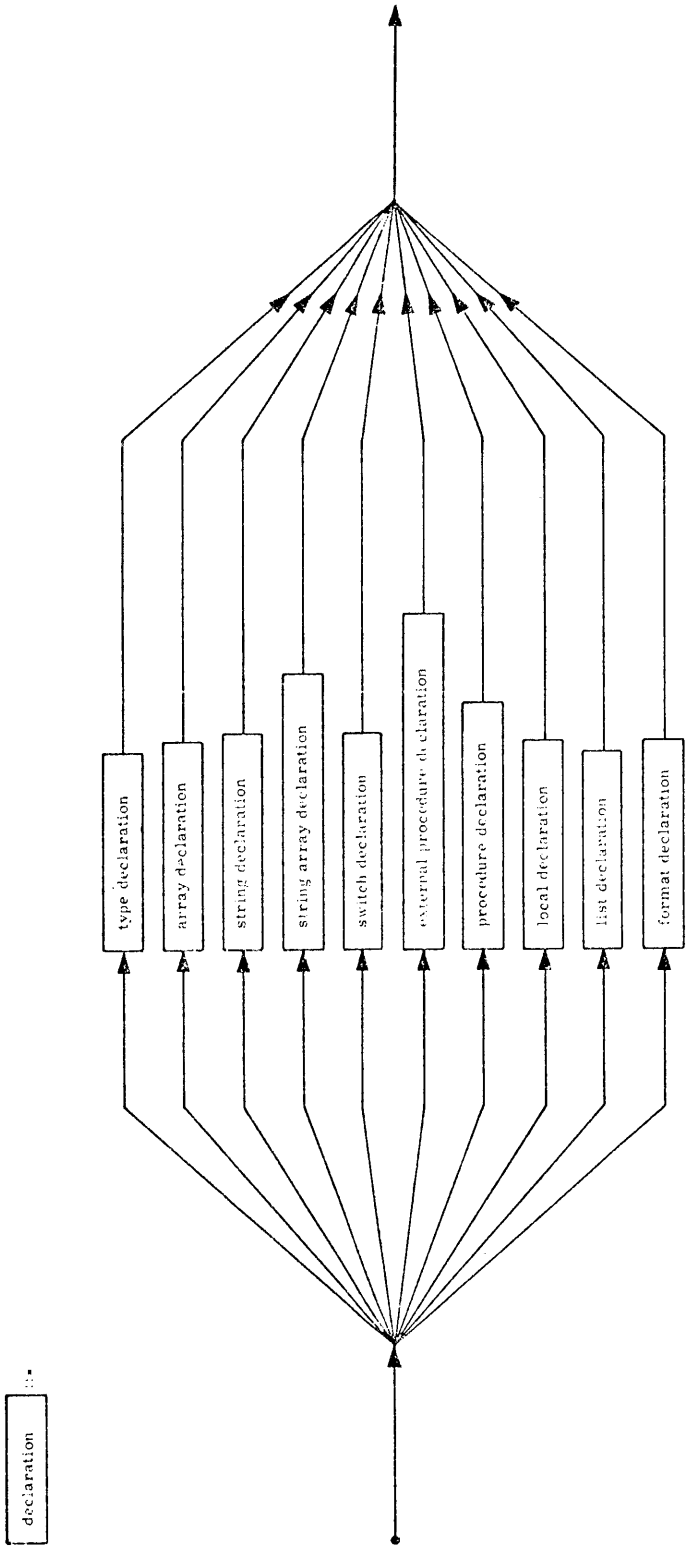
List of Charts



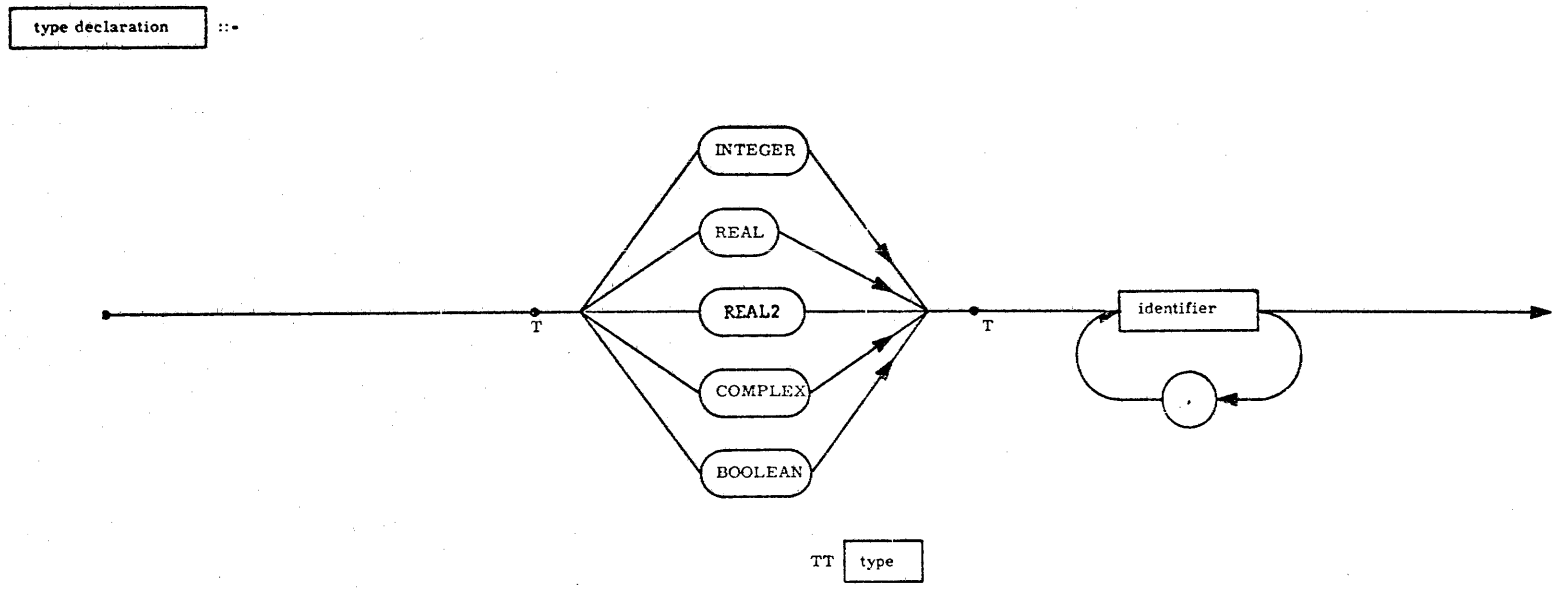
Explanation: A program is a complete set of declarations and statements which define an algorithm for solving a problem. The logic of this algorithm (its correctness) is the business of the programmer. The compiler only checks that the syntax (form) is correct.

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.



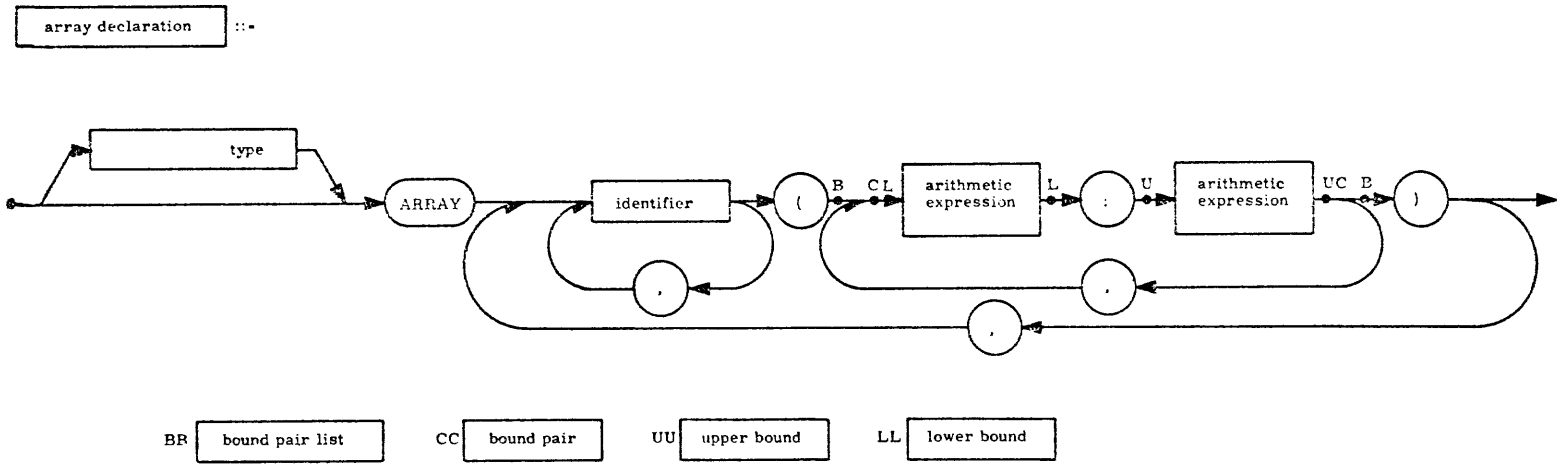
Explanation: There are 10 types of declarations each of which is defined in detail on the following pages.



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.

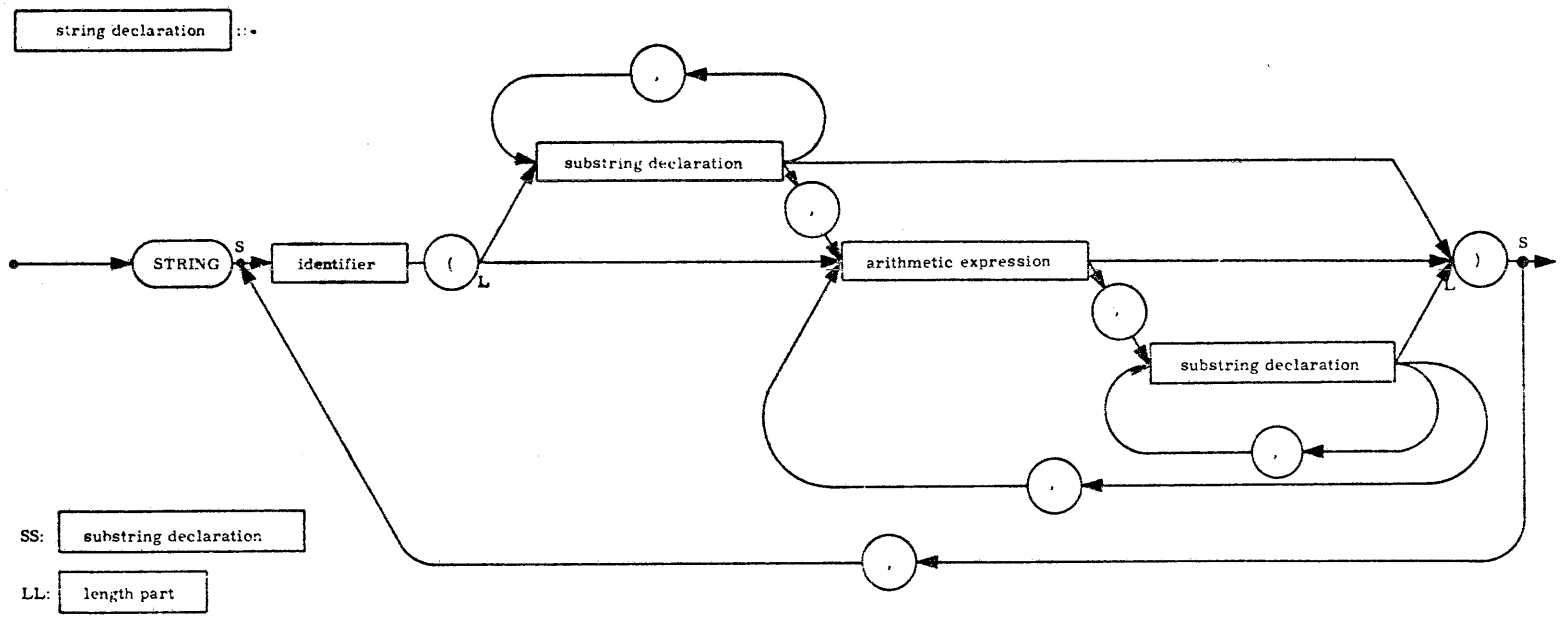
Examples:

```
INTEGER I4, PAK, LOOPCNT $
     BOOLEAN ANYLEFT, LASTOUT $
COMPLEX C, CINVS $
REAL2 DP $
     REAL  QIN, QOUT, MAXITEM $
```



Explanation: 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.

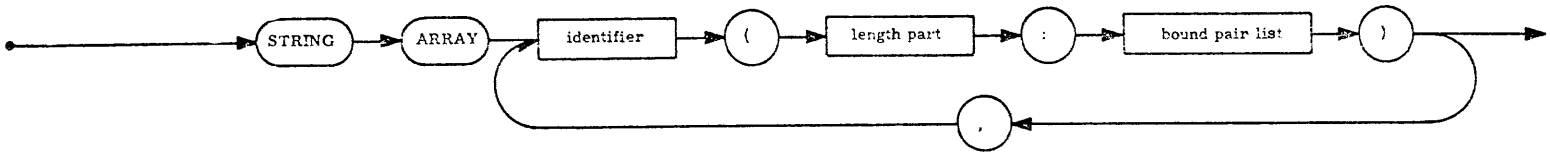
Examples:
```
COMPLEX ARRAY  CC0N4 (0:N), CP1(1:N+1) $
BOOLEAN ARRAY BAND, BOR, BXOR(-4:4) $
REAL ARRAY     B(I-1:I+1), XINITIAL, YINITIAL(-N:N, -N:N, 1:2) $
    INTEGER ARRAY I(1:5), J, K, L(ENTIER(X): P112) $
ARRAY XYZ4(1:N×2) $
```



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.

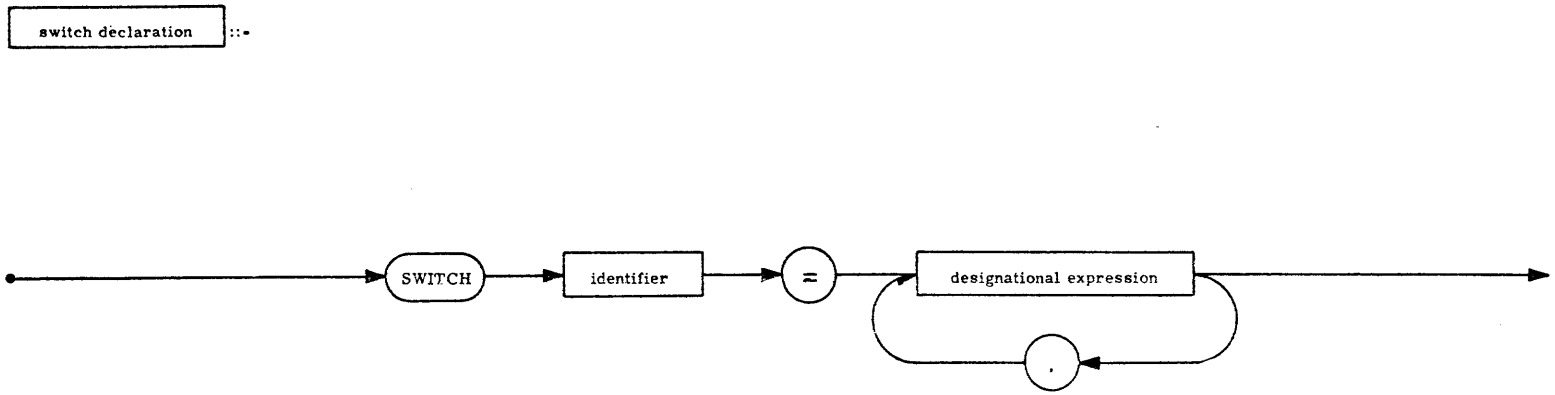
Examples:
```
STRING ST1(36), NAME(INITIALS(2), LAST(16)) $
STRING PI(N+2), QUOTE(1) $
    STRING NEXTOUT(80) $
STRING ALPHA(BETA(2, GAMMA(4), 2), DELTA(EPSILON(6)), 20) $
```

string array declaration ::=



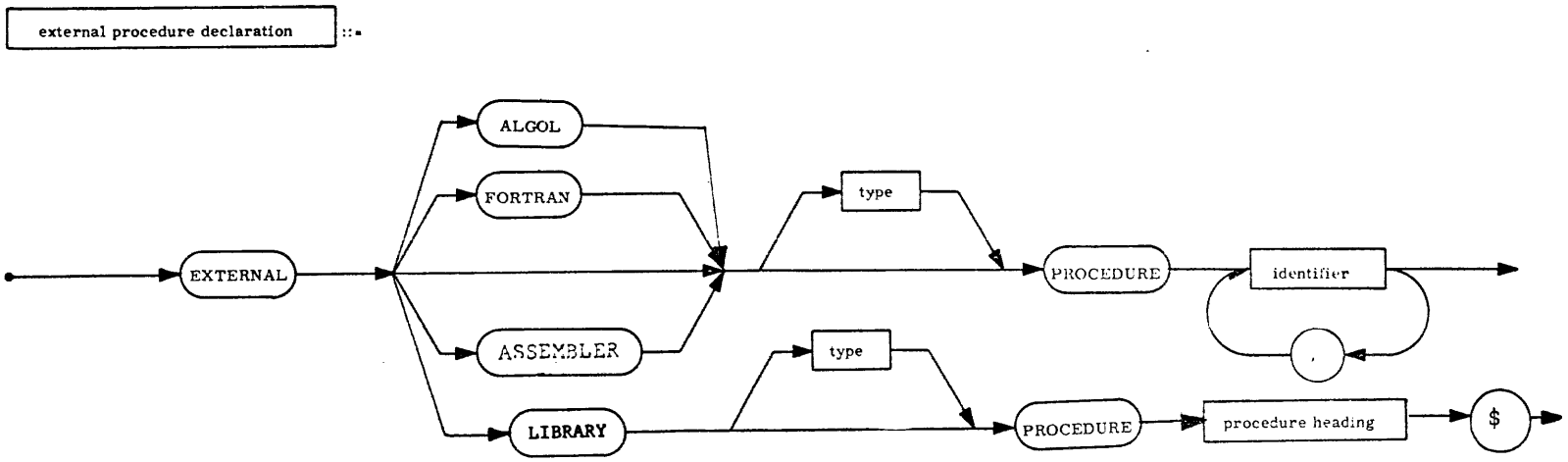
Explanation: A string array is a matrix whose elements are strings. Appended to the length part of the declaration are the bound pairs for each dimension, just as for an ordinary array.

Examples:
```
STRING ARRAY SA(80:0:100),CARD(LABEL(8),OP(6),2,OPERAND(64):1:N) $
    STRING ARRAY LASTFILE (CLENGTH:1:507) $
```



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.

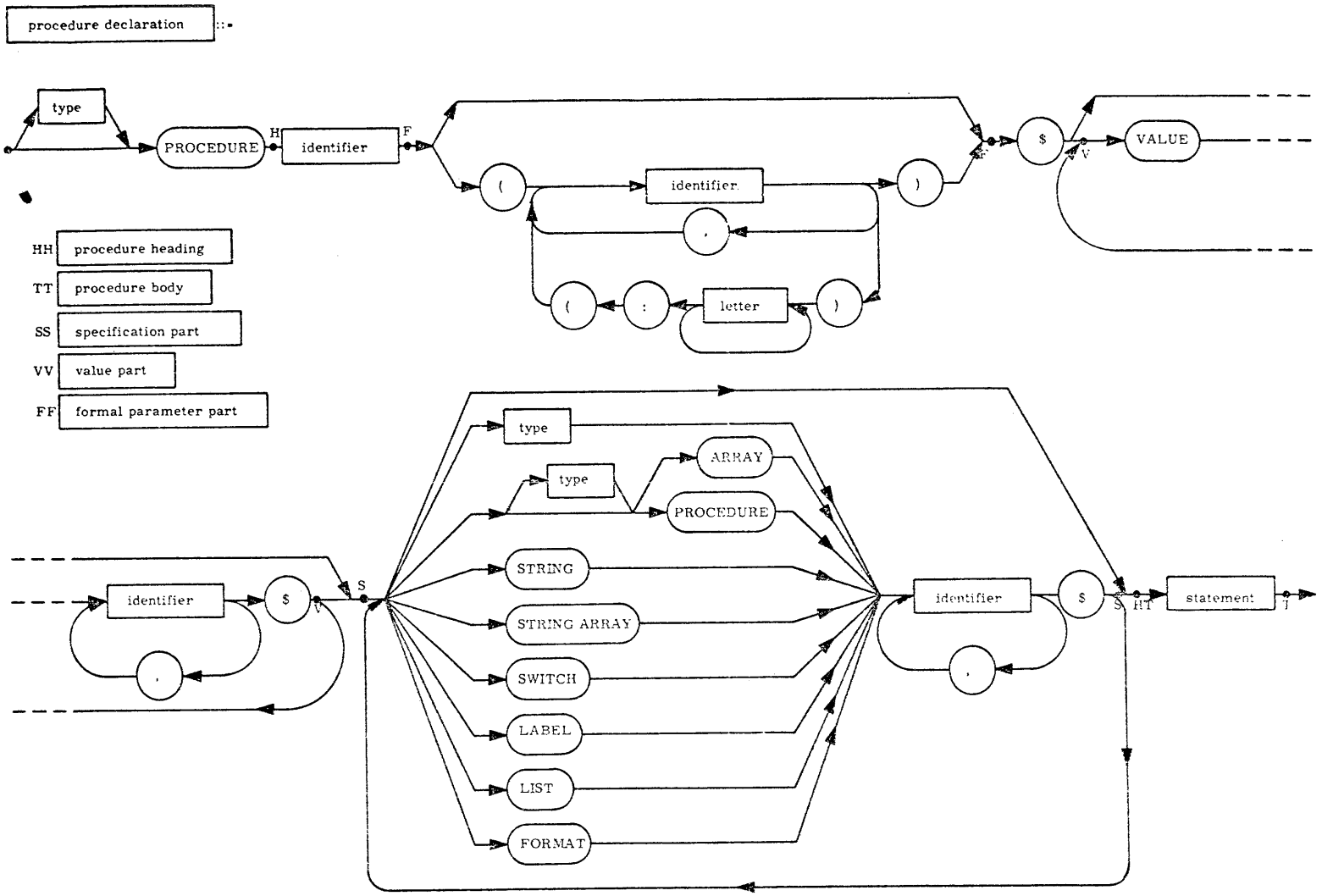
Examples:
```
SWITCH JUMP = L1,START,FEIL4,SLUTT $
SWITCH BRANCH = IF BETA EQL 0 THEN L1 ELSE JUMP(J),START $
```



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 (ASSEMBLER, LIBRARY), ALGOL or FORTRAN. The type of external procedures is specified if they are functional procedures.

Examples:

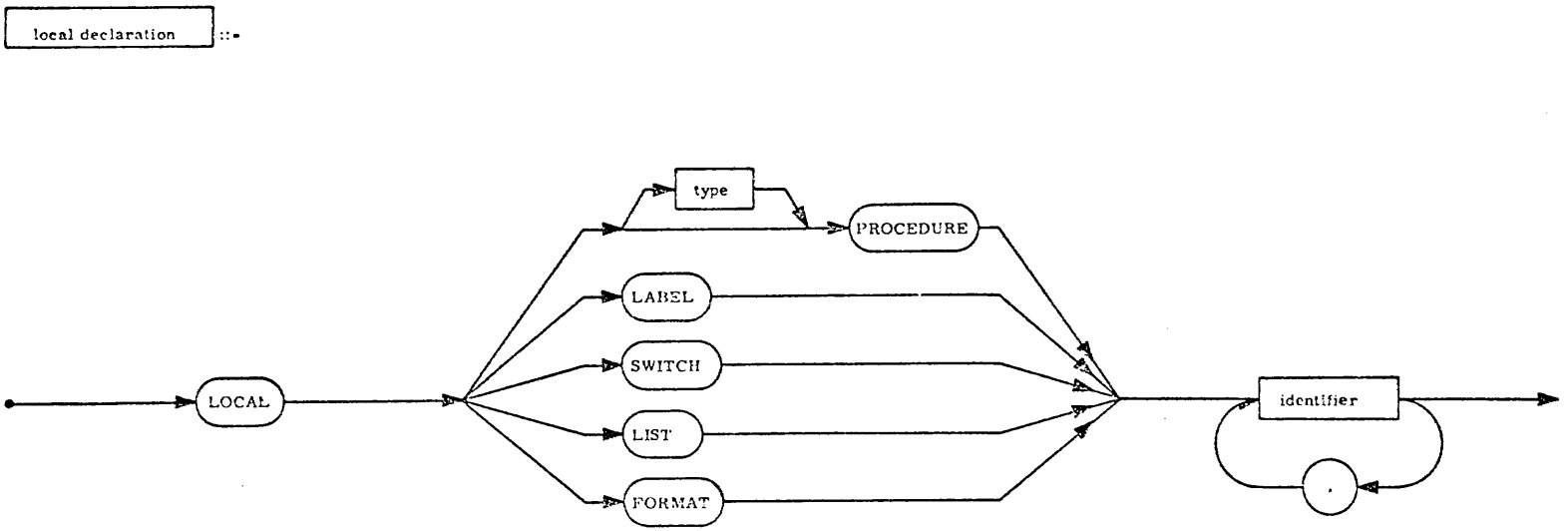
```
EXTERNAL FORTRAN REAL PROCEDURE CBRT$
EXTERNAL FORTRAN PROCEDURE NTRAN,INVS$
EXTERNAL PROCEDURE ROOTFINDER,KEYIN,KEYOUT$
EXTERNAL ASSEMBLER PROCEDURE TYPEIN,TYPEOUT$
EXTERNAL LIBRARY INTEGER PROCEDURE PACK(A,B,C)$
                                VALUE A;B$
                                INTEGER A,B,C$ $
```



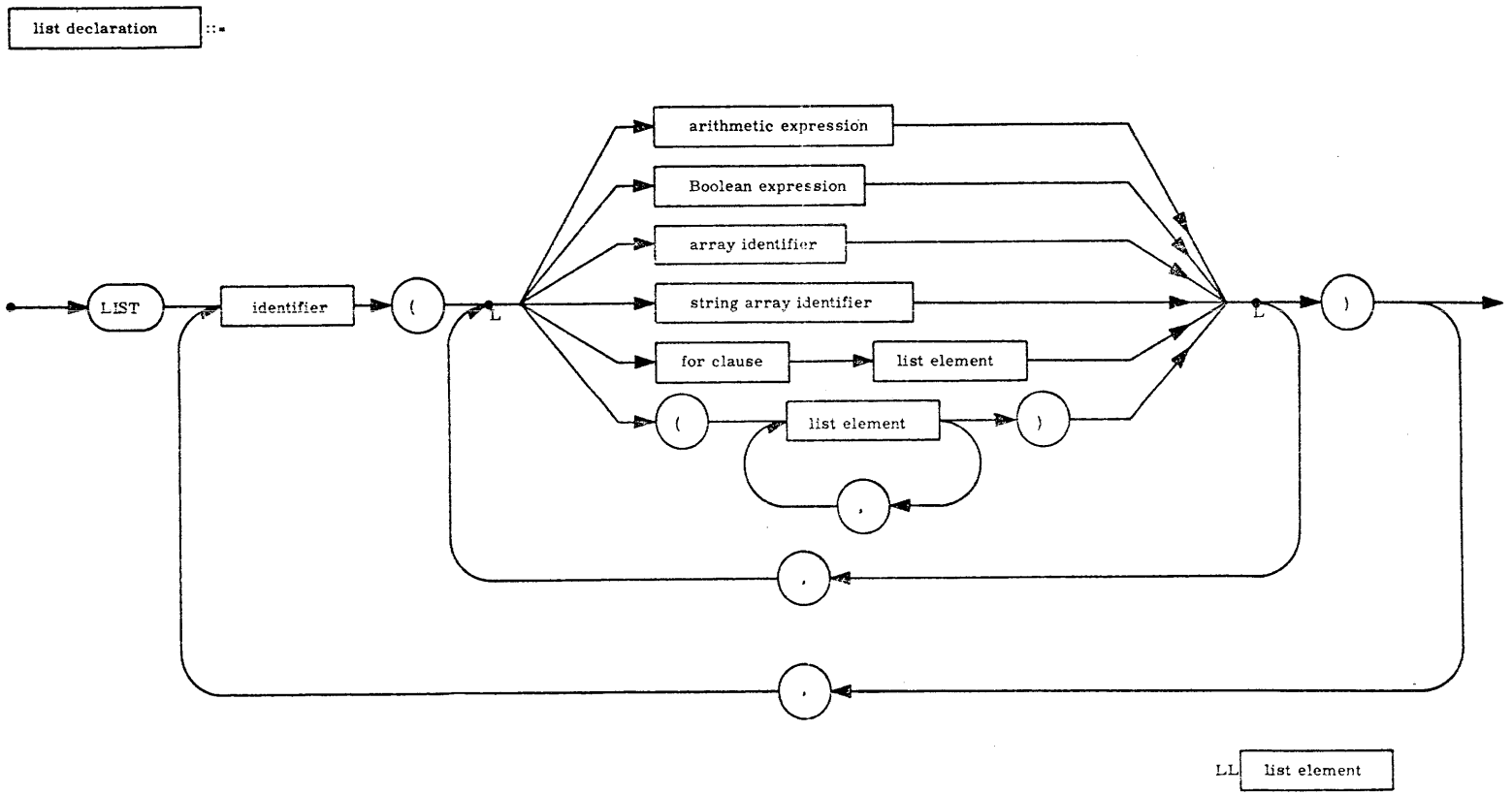
Explanation: A procedure declaration defines an algorithm to be associated with a procedure identifier. The principal constituent of a procedure declaration is a statement which is executed when the procedure is "called" (see procedure statement and function designator). The procedure heading specifies 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 OR FIRST AND LAST) $
```

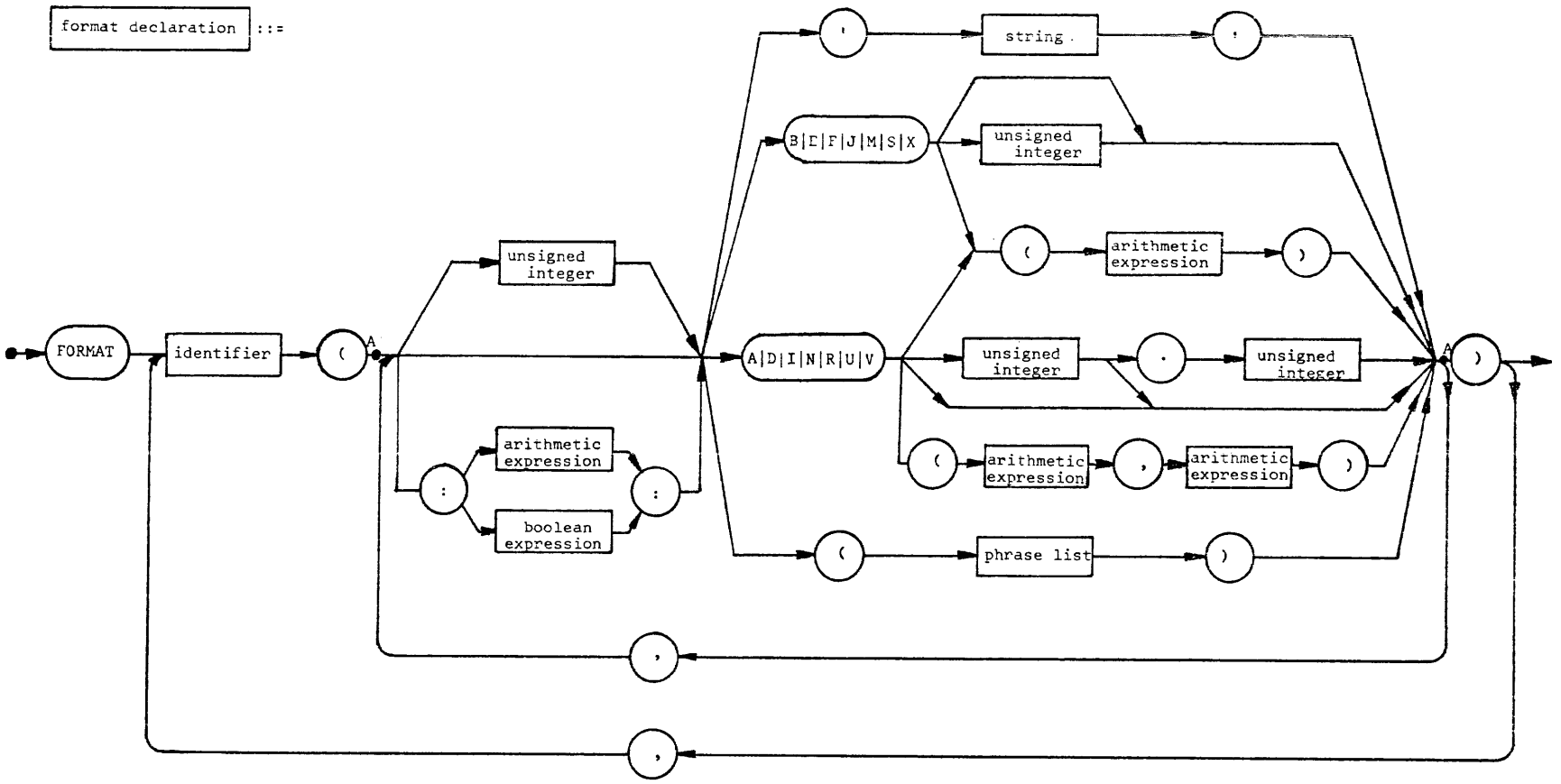


Explanation: The local declaration in NU ALGOL is treated as a dummy declaration and has been retained only for compatibility with the with the UNIVAC 1106/1108 ALGOL.



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) 5
```

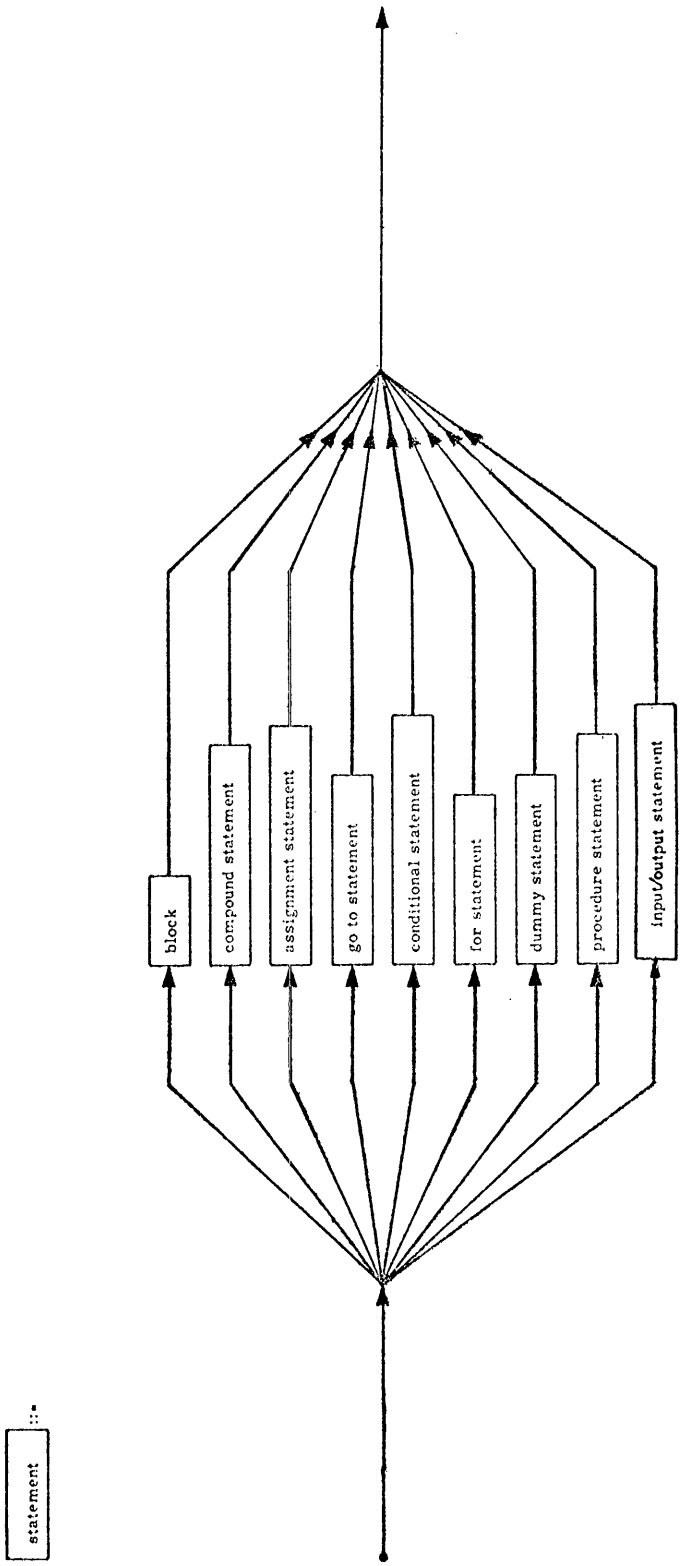


AA phrase list

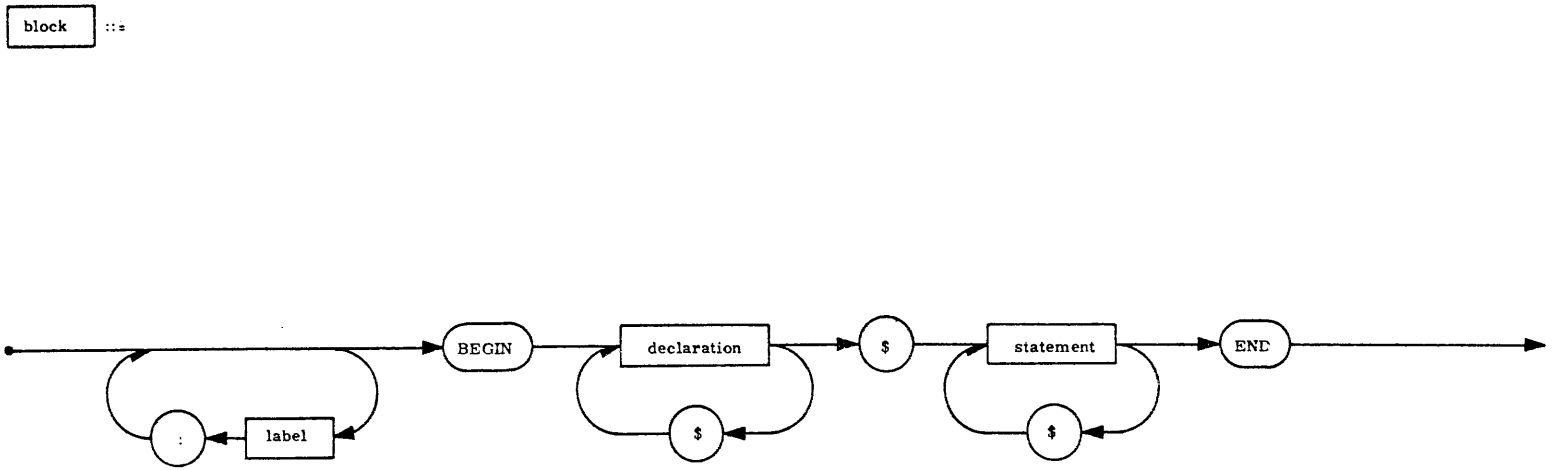
Explanation: A format is a special string of symbols which are passed on to an input/output routine for editing and control. Integers in front of a format code specify the number of times that code is to be repeated.

Examples:

```
FORMAT NEWPAGE(E,'X-COORDINATE',X28,'Y-COORDINATE',A1) $
FORMAT REP(5(4 R16.8,A1),A0.2,S12,'=',D10.1,S12,'=',D10.1,A1) $
FORMAT VECTOR (10T10.4,A1),PATTERN('SWITCHES ARE',886,A1) $
FORMAT MATRIX (:N:(:M:(D4.2,A1))) $
```



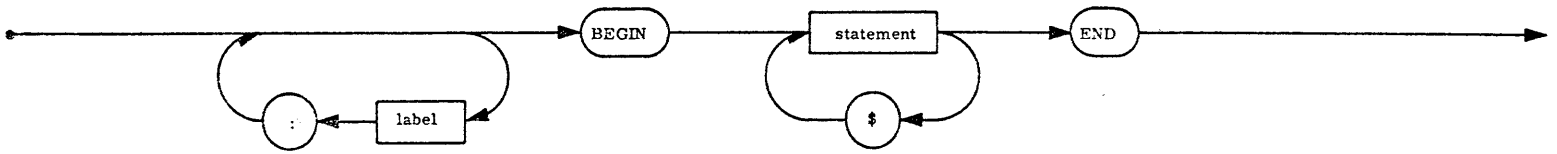
Explanation: Statements define the sequence of operations to be performed by the program. The 9 types of statements are each defined in the following pages.



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).

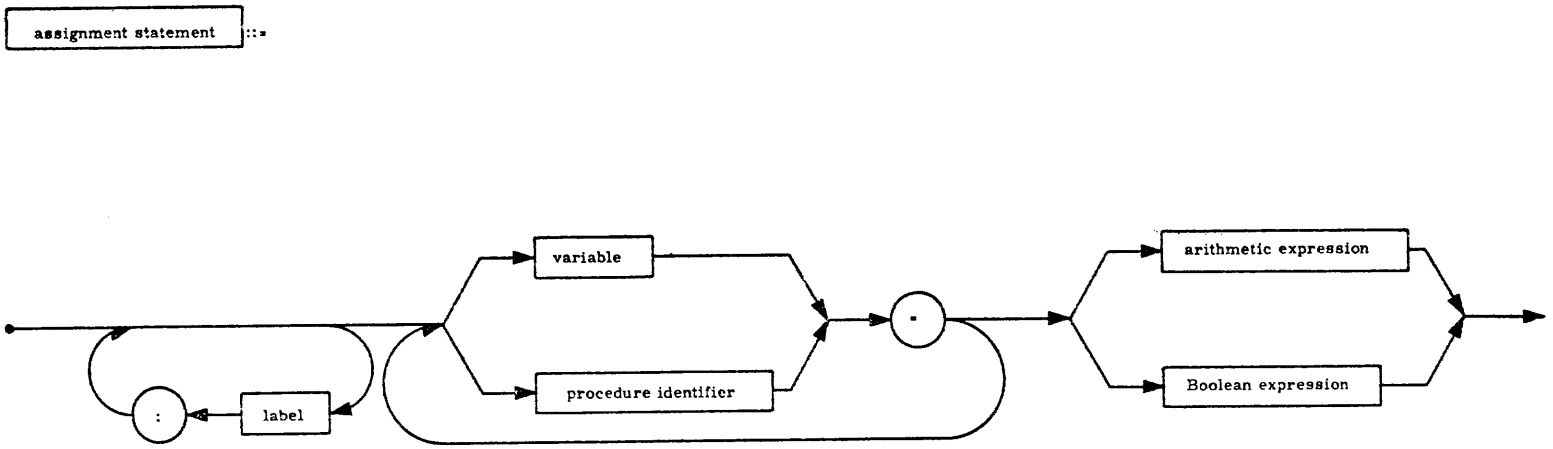
Example:
```
L:BEGIN  INTEGER ARRAY A(1:10) $
         A(1) = 1 $
         FOR J = (2,1,10) DO A(J) = A(J-1) + J $
         FOR J = (1,1,10) DO WRITE (J, A(J)) $

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.

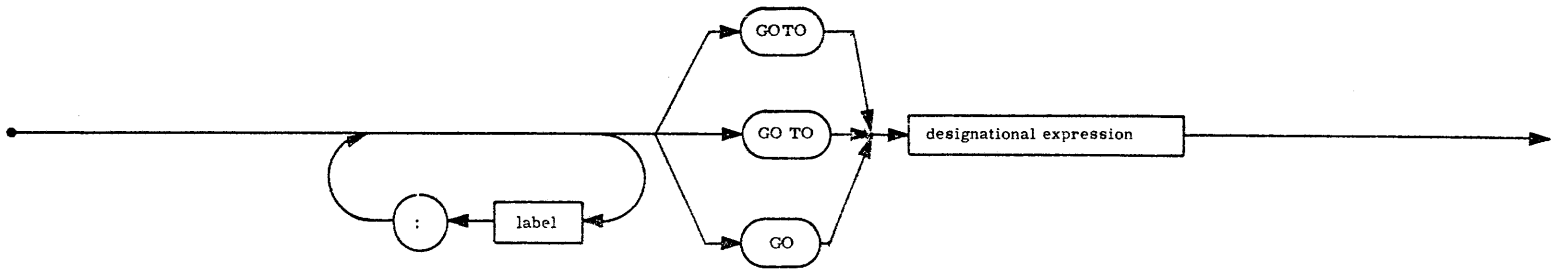
Example:
```
BEGIN  T= 0 $ FOR I = 1 STEP 1 UNTIL M DO
           T= B(J,I) x C(I,K) + T $
           IF T GTR 820 OR OVFLOW THEN GO TO SPILL $
END $
```



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.

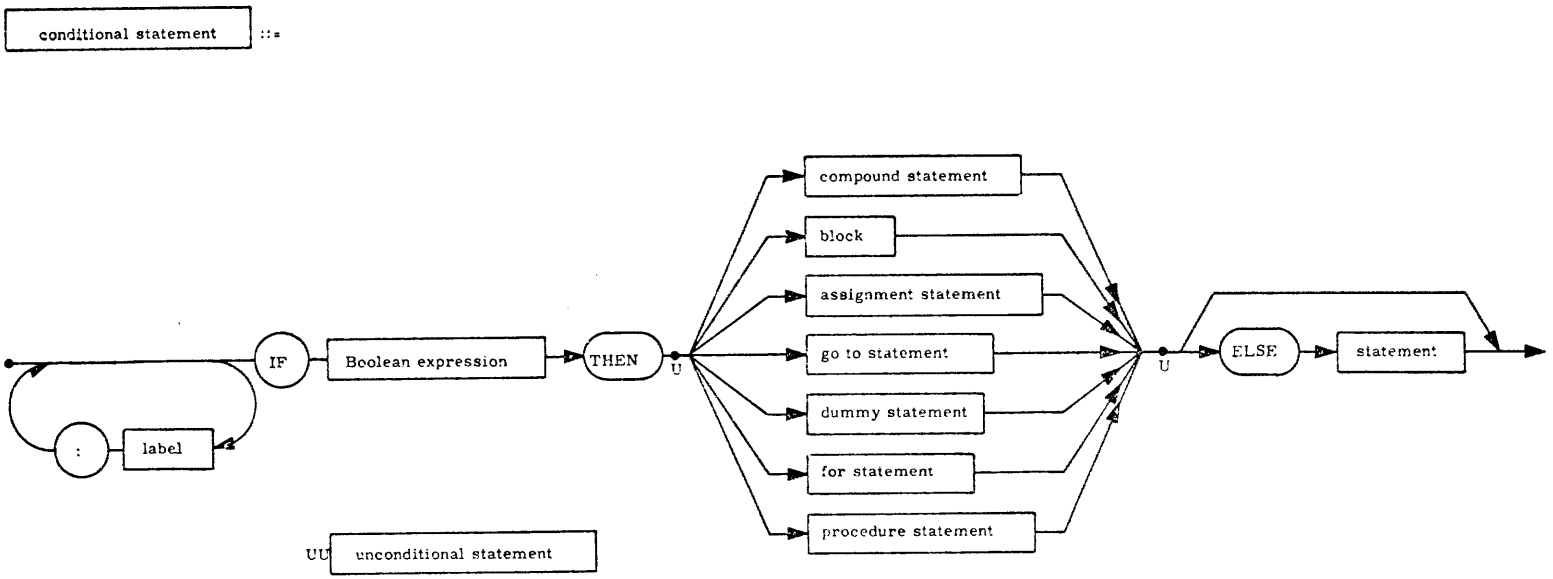
Examples:
```
A(I) = B(I) = &35 $
AANDB = A AND B OR EPS1 GEQ EPS2 $
P = SQRT(B**2 - 4*A*C) $
T = S - MYO*EPSO*(2*PI*F)**2$
S(V,K-2) = COS(ANGLE) + 0.5 *(IF S1 THEN K**3 ELSE K**5) $
NAME(1,6:P + 1) = 'IFTHEN' $
```

go to statement ::=



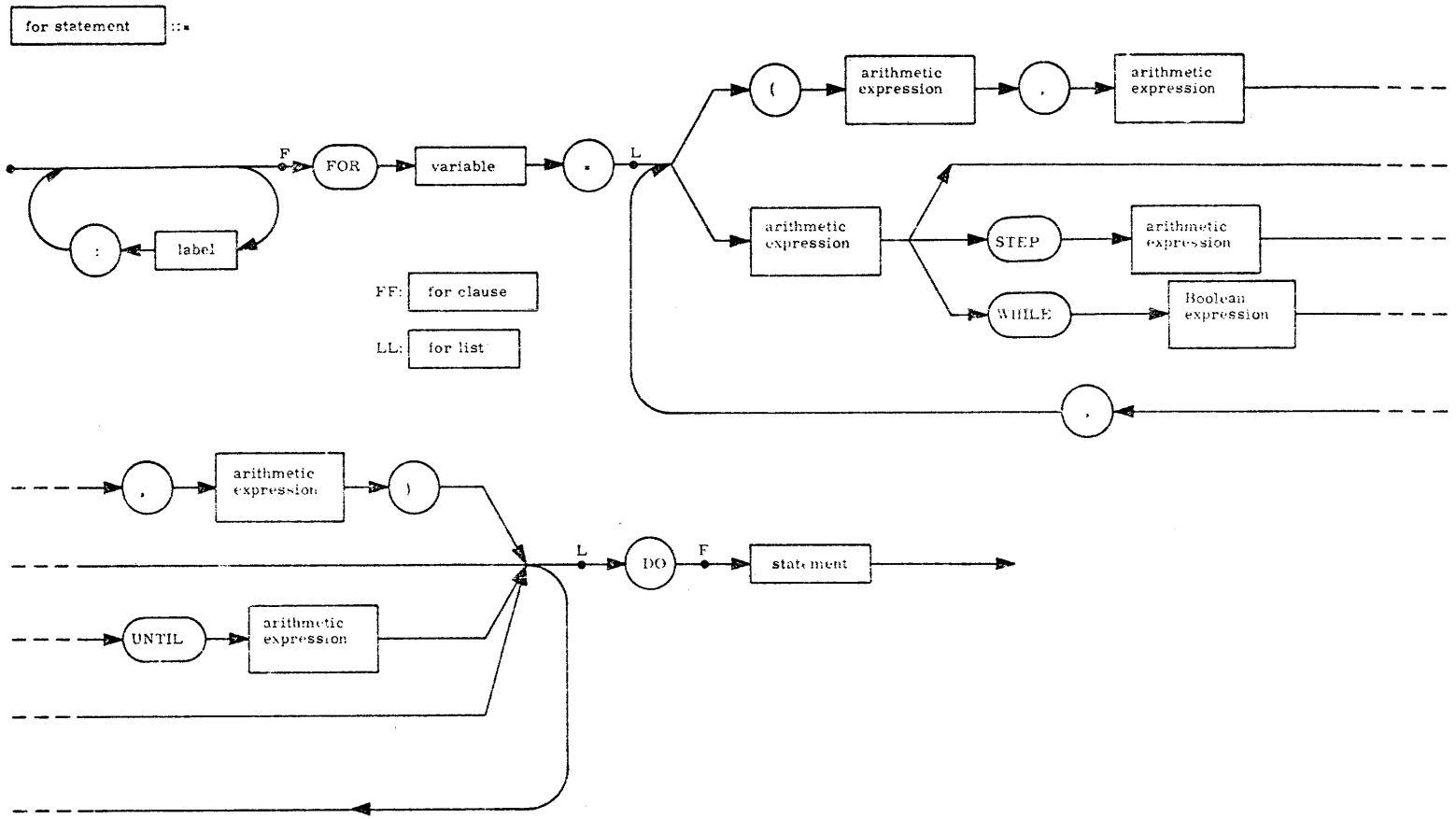
Explanation: A go to statement causes transfer of control to the statement with the label determined by the designational expression.

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)) $
```



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 $
```

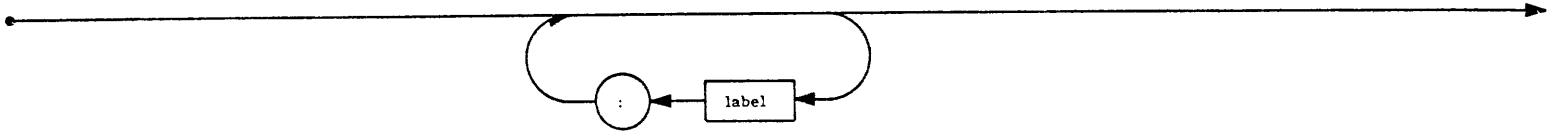


Explanation: The FOR statement controls the execution of the statement following the DO a number of times while the variable to the left of the = is assigned the values determined by the for list. The (,,) construction is equivalent to the STEP-UNTIL construction.

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 $
```

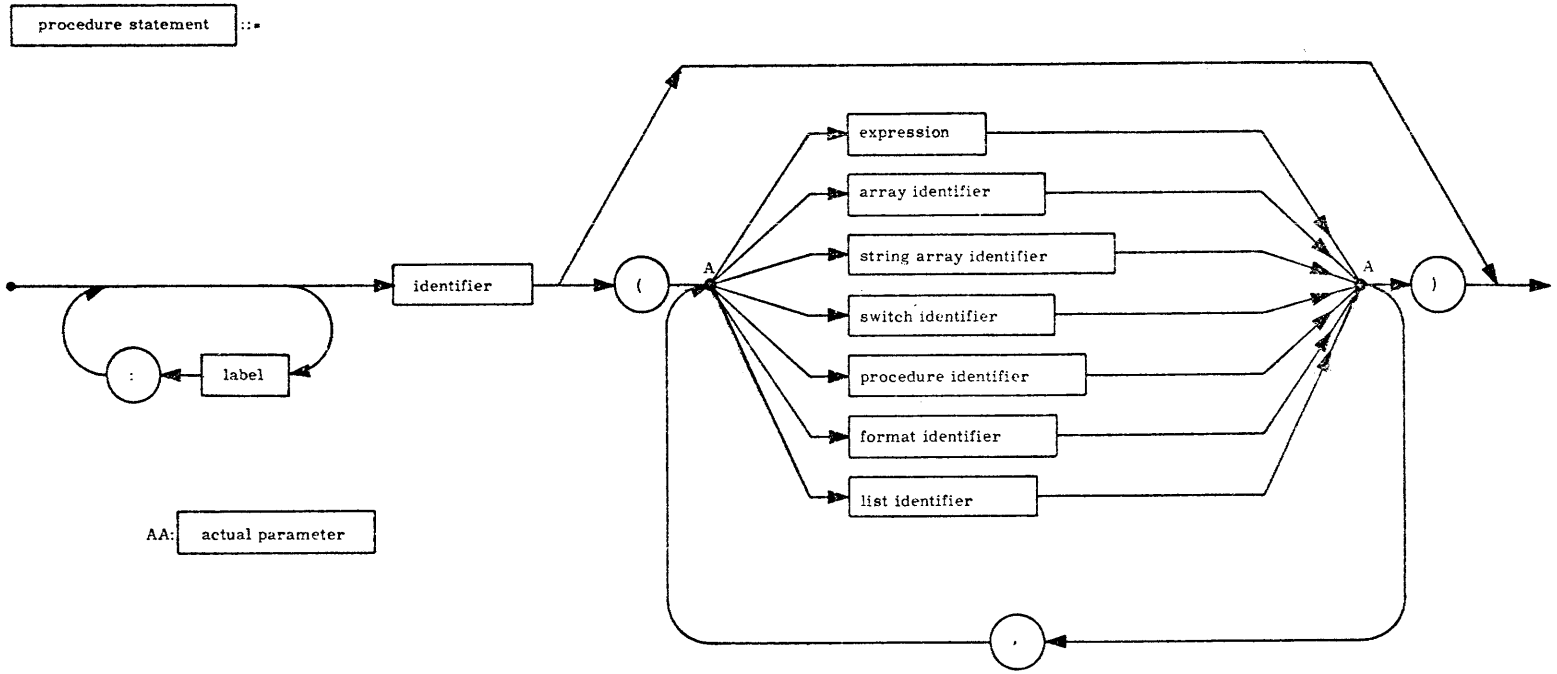
dummy statement ::-



Explanation: A dummy statement does nothing. It may serve to place a label.

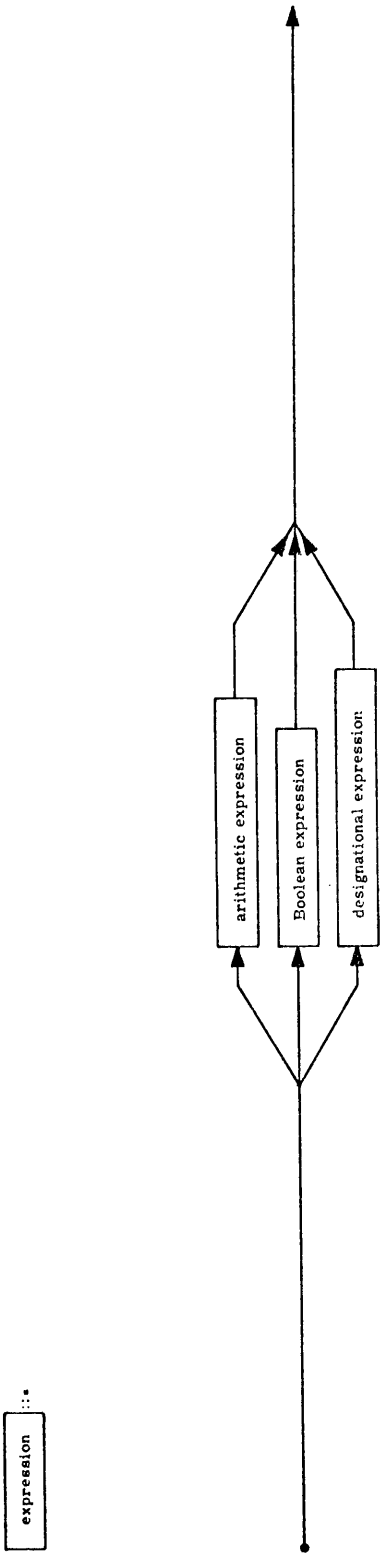
Examples:

```
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 $
```



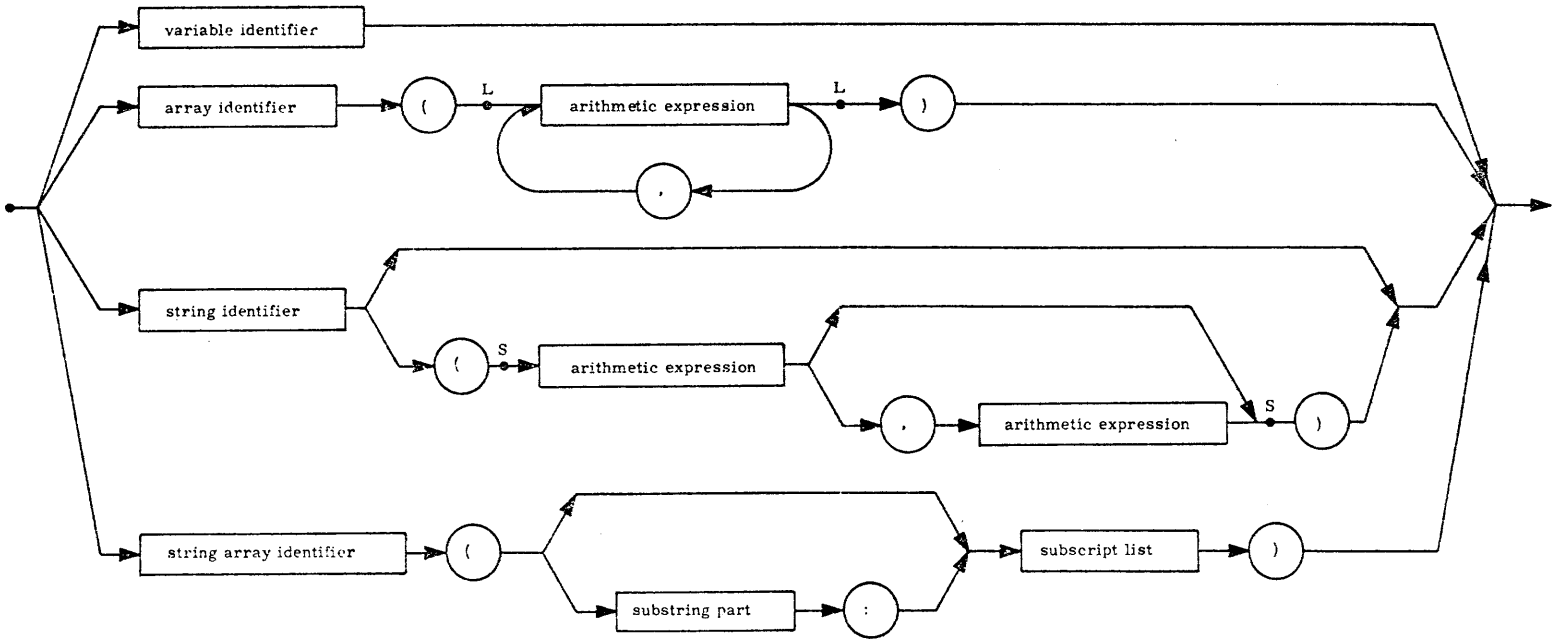
Explanation: A procedure statement is a call on a declared procedure. The actual parameters of the call replace the formal or dummy parameters throughout the body of the declared procedure. If the corresponding formal parameter has been "VALUE" specified then only the value of the actual parameter is 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) $
```



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 ::=

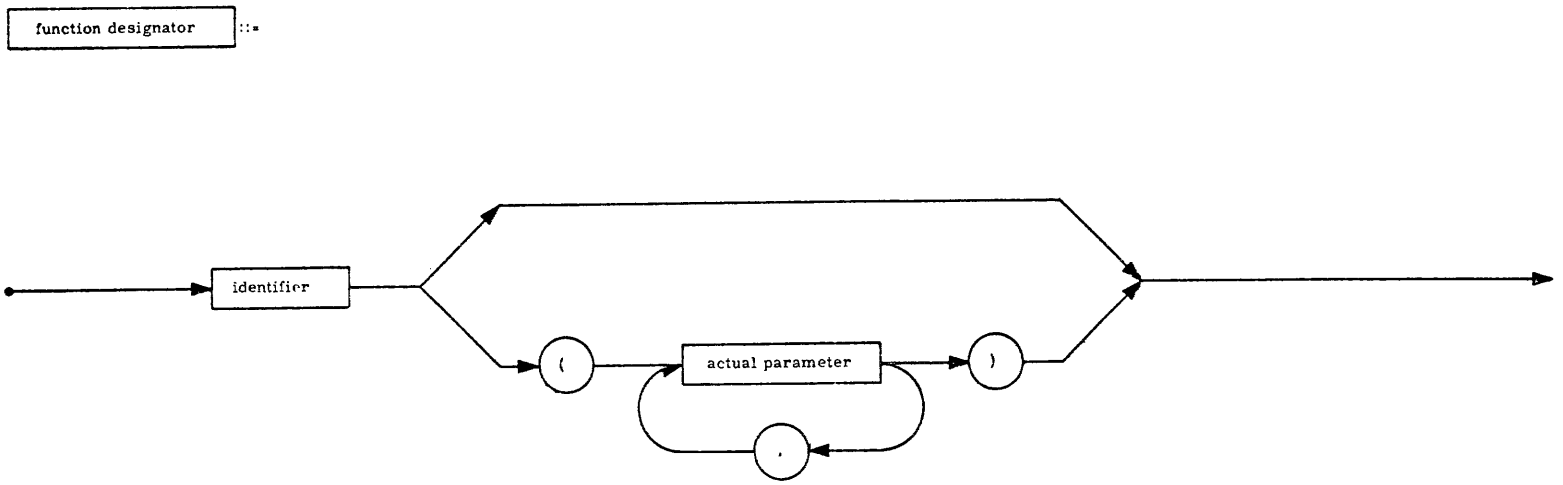


LL: subscript list

SS: substring part

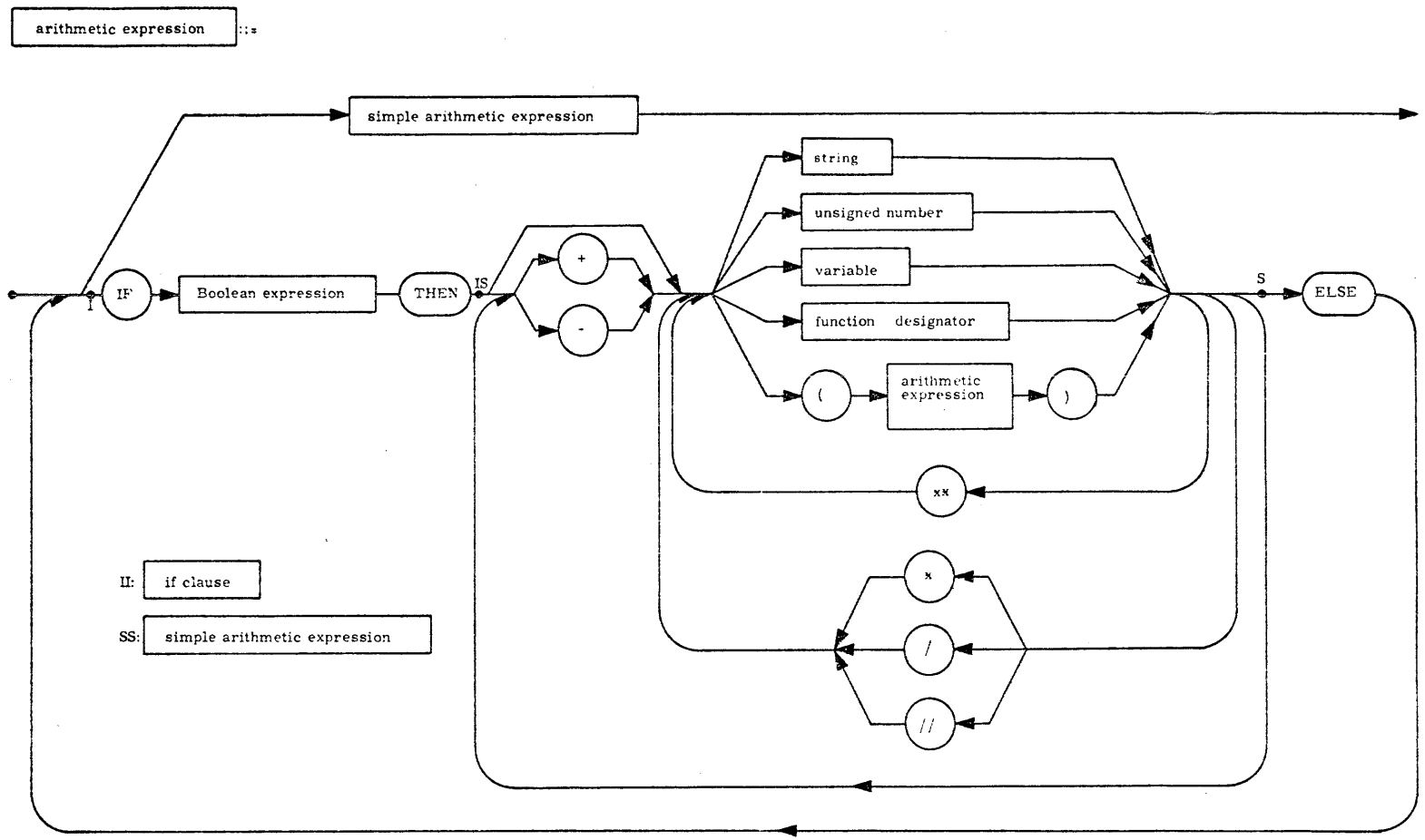
Explanation: A variable is a designation given to a single value. A variable identifier is a variable named in a type declaration.

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)



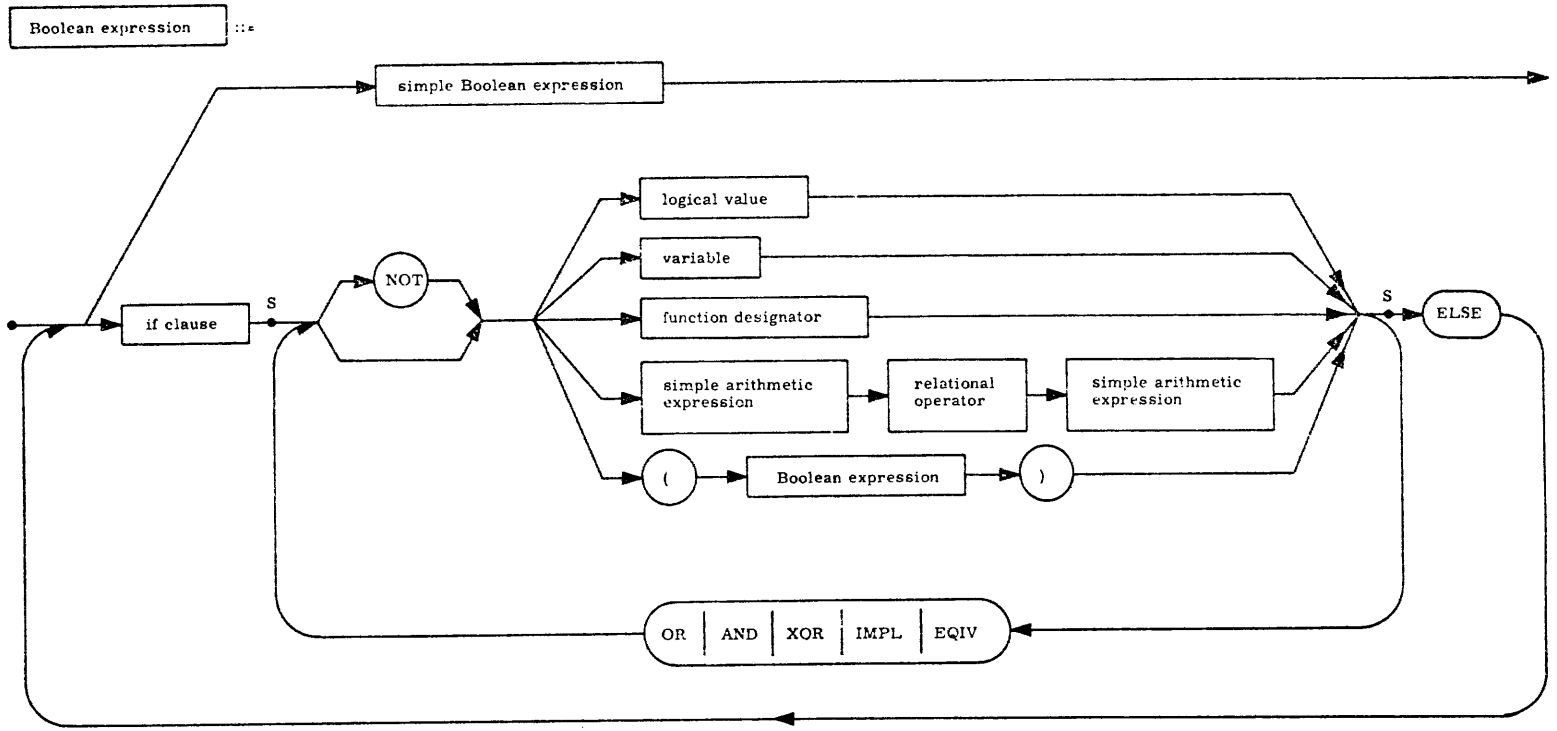
Explanation: A function designator defines a single numeric or logical value by applying the rules of the procedure declaration to the actual parameters. Only a procedure which has a type associated with it can be a function designator. Besides those functional procedures declared in the program, several standard ones are available for use without being declared.

Examples:
CLOCK
ARCTAN(1.0)
BACKSLASH(A1, A2)



Explanation: An arithmetic expression is a rule for computing a numerical value.

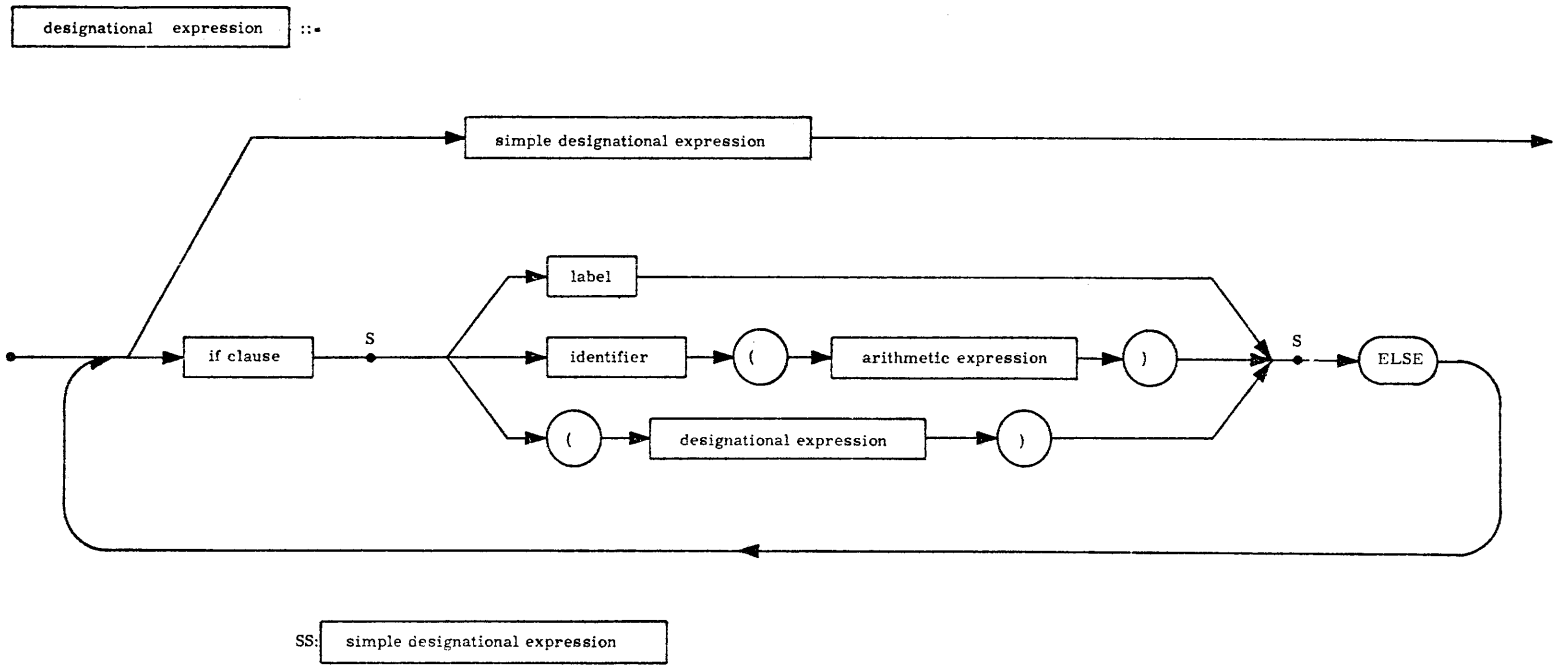
Examples:
A(4) + 2 * SQRT(D**3) - DELTA
IF A LSS &-5 THEN 0 ELSE A/&5
Q(MOD(N, 2) + 1) * (IF FIRST THEN 10 ELSE RATIO) // 3



SS: simple Boolean expression

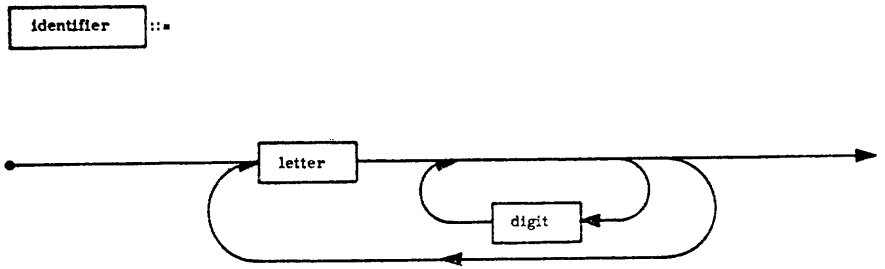
Explanation: A Boolean expression is a rule for computing a logical value.

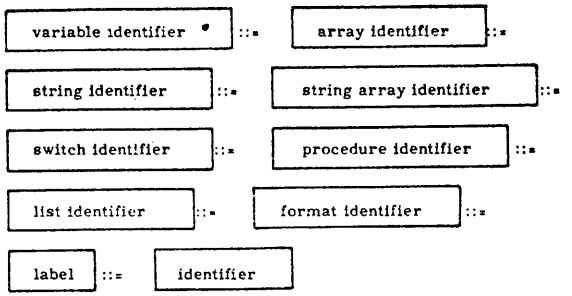
Examples: FIRST AND NOT SPECIAL
A LSS DELTA OR ITERATIONS GTR MAXN
IF BETA THEN TRUE ELSE IF STEP(I) IMPL STEP(I+1) THEN QVALUE(P,I) ELSE QVALUE(P,I-1)

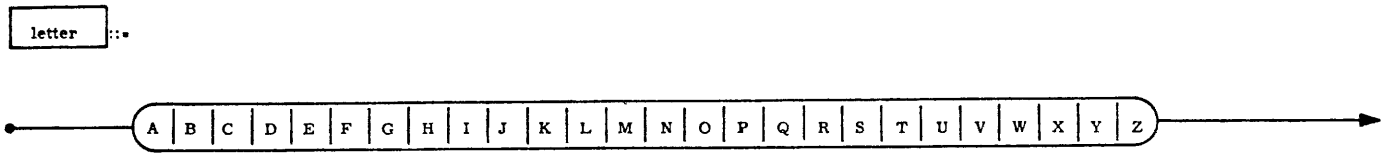


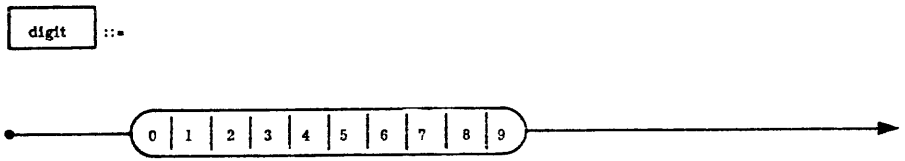
Explanation: A designational expression is a rule for computing the label of a statement. A switch identifier followed by an arithmetic expression in parenthesis refers to the label in the corresponding position in the switch declaration.

Examples:
```
L10
IF BETA THEN SLUTT ELSE NEXT (K//2)
```



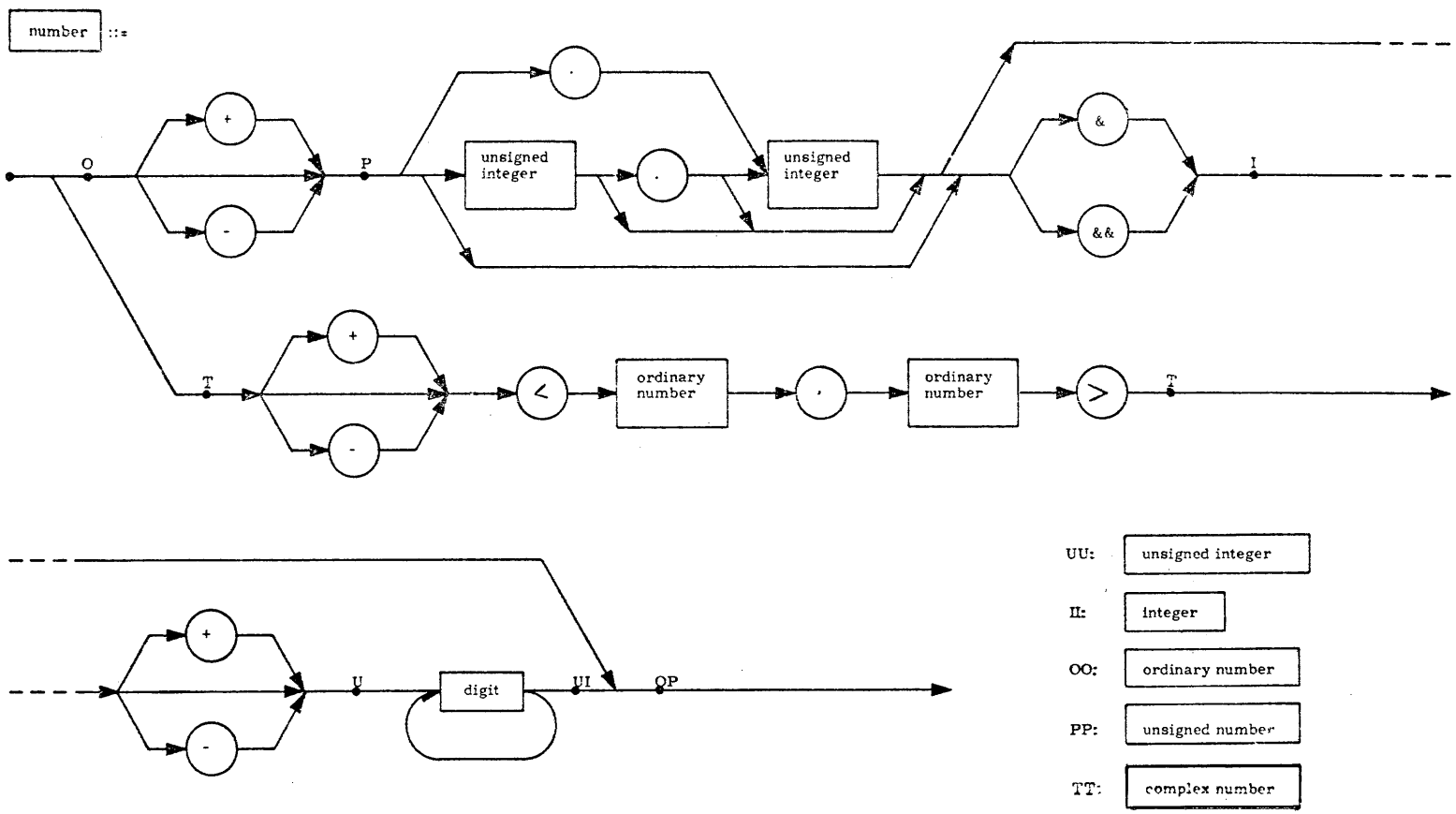






Explanation: An identifier is a name chosen to represent a variable, array, etc. Only the first 12 characters of an identifier uniquely define it.

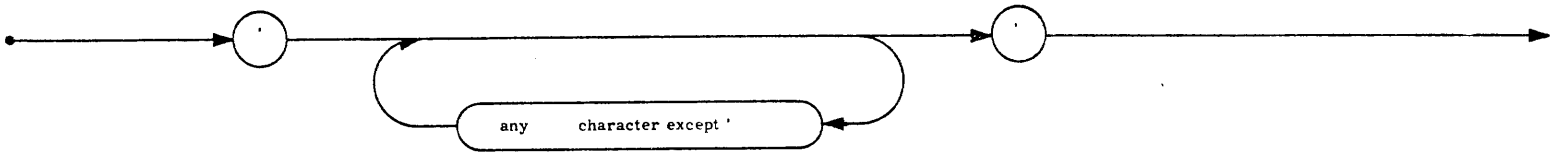
Examples: P47
DELTA
SQRTROOOF2
E1C4PDQ



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 18 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)

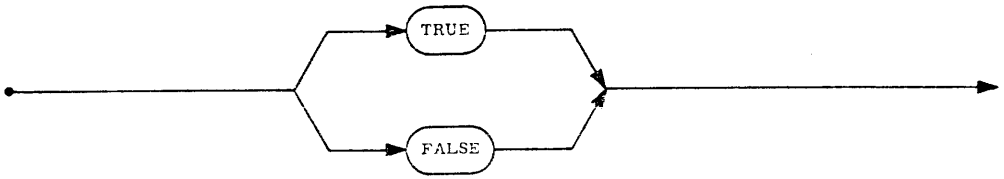
string ::=



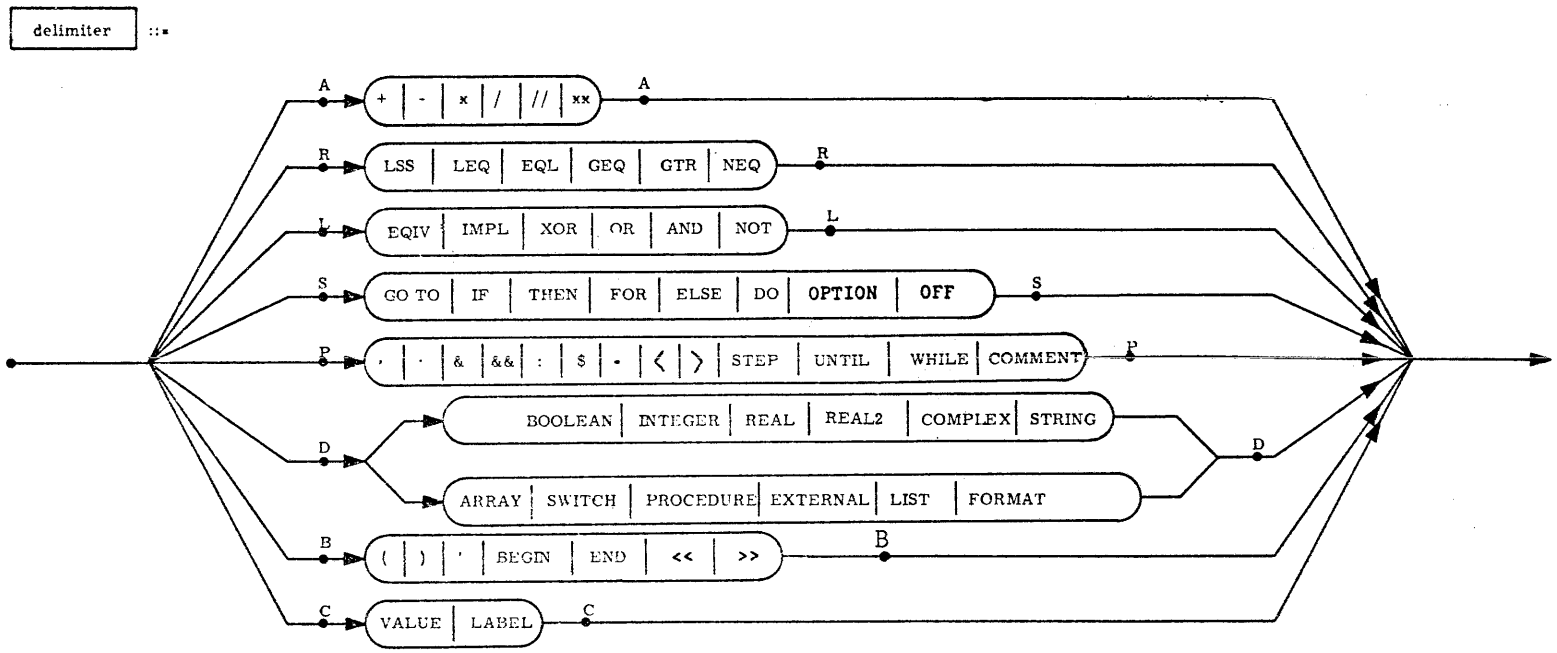
Explanation: A string constant is any string of characters which are used as parameters to procedures or with string variables.

Examples:
'DOGGENBURG STR. 22'
'NEQ'
'BJARNE WIST'
'227 KALPHA'
' REAL ARRAY '

logical value ::=

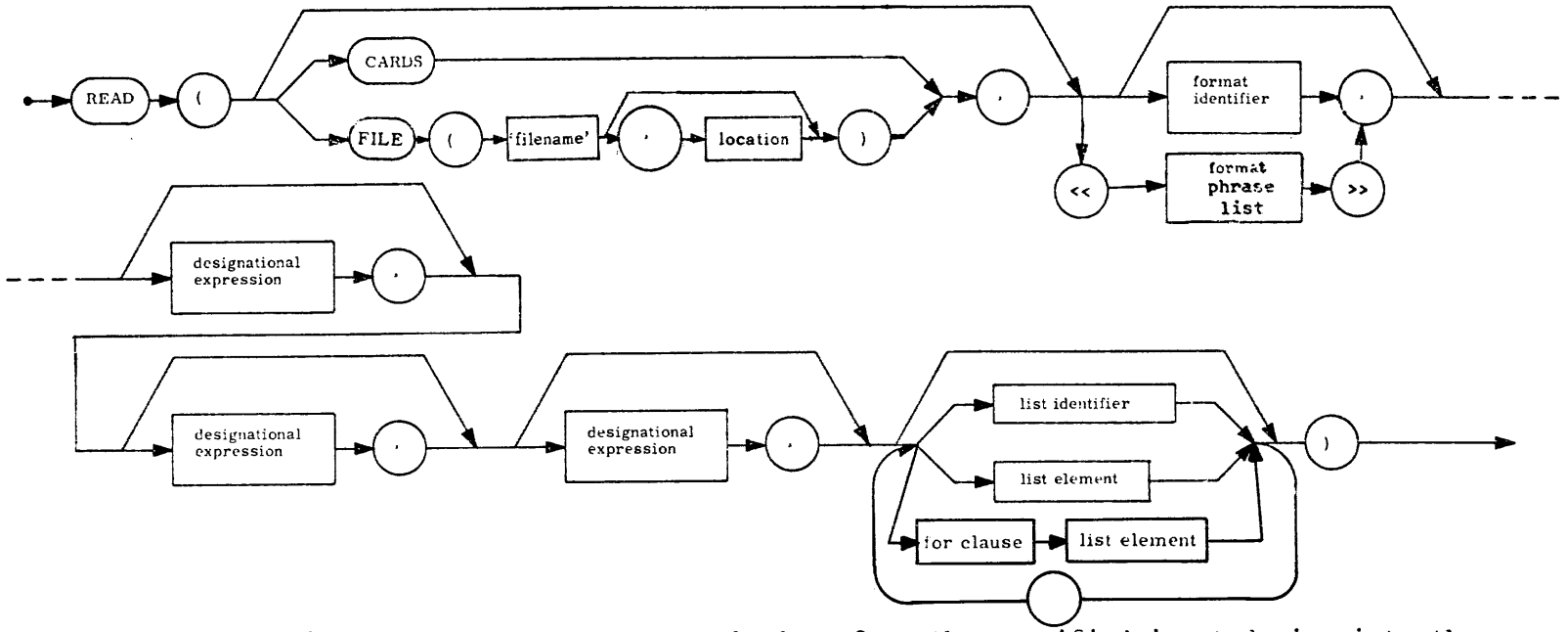


Explanation: A logical value is a Boolean constant.



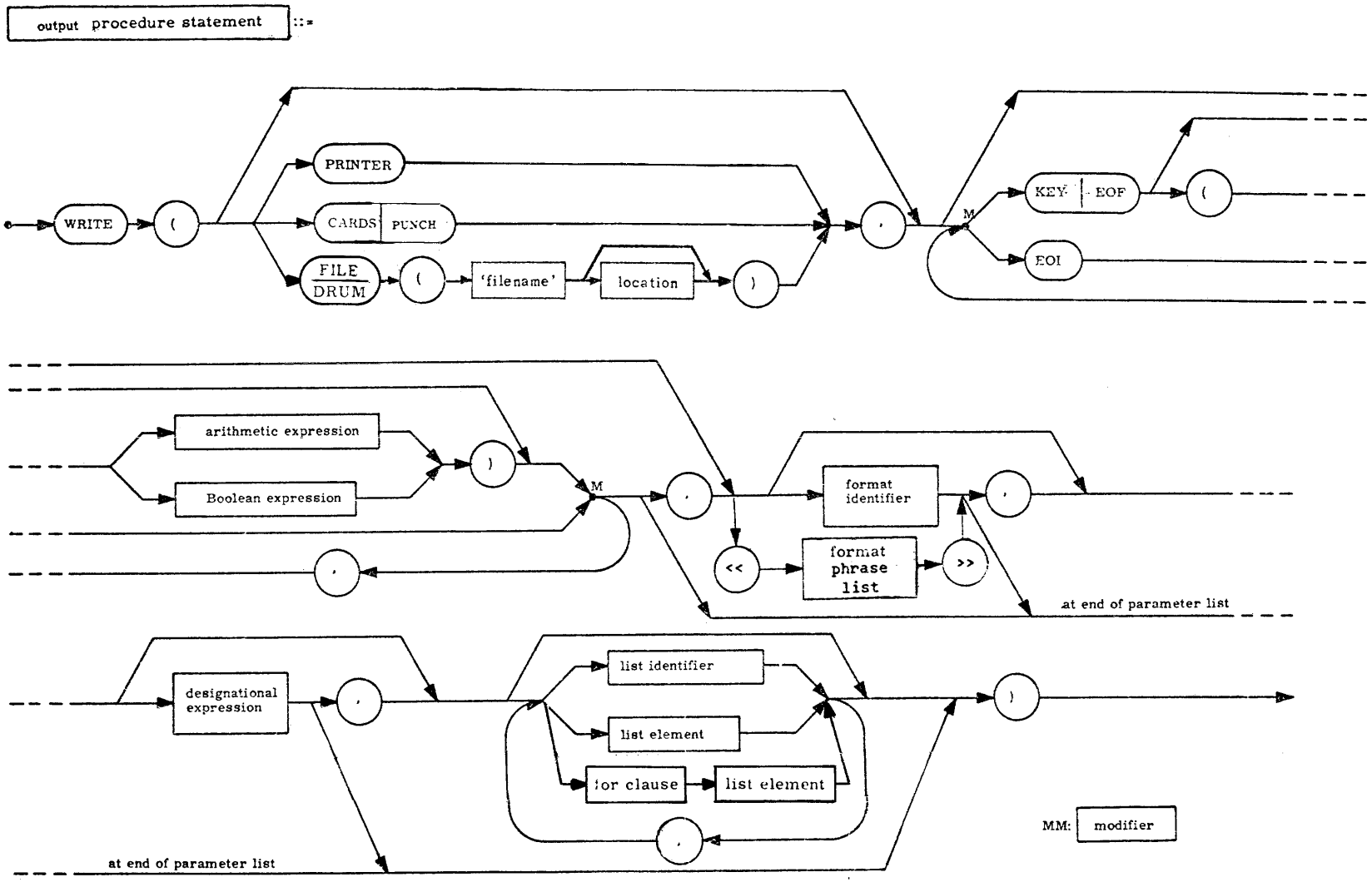
AA:	arithmetic operator	PP:	separator
RR:	relational operator	DD:	declarator
LL:	Boolean operator	BB:	bracket
SS:	sequential operator	CC:	specificator

input procedure statement ::=



Explanation: 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.

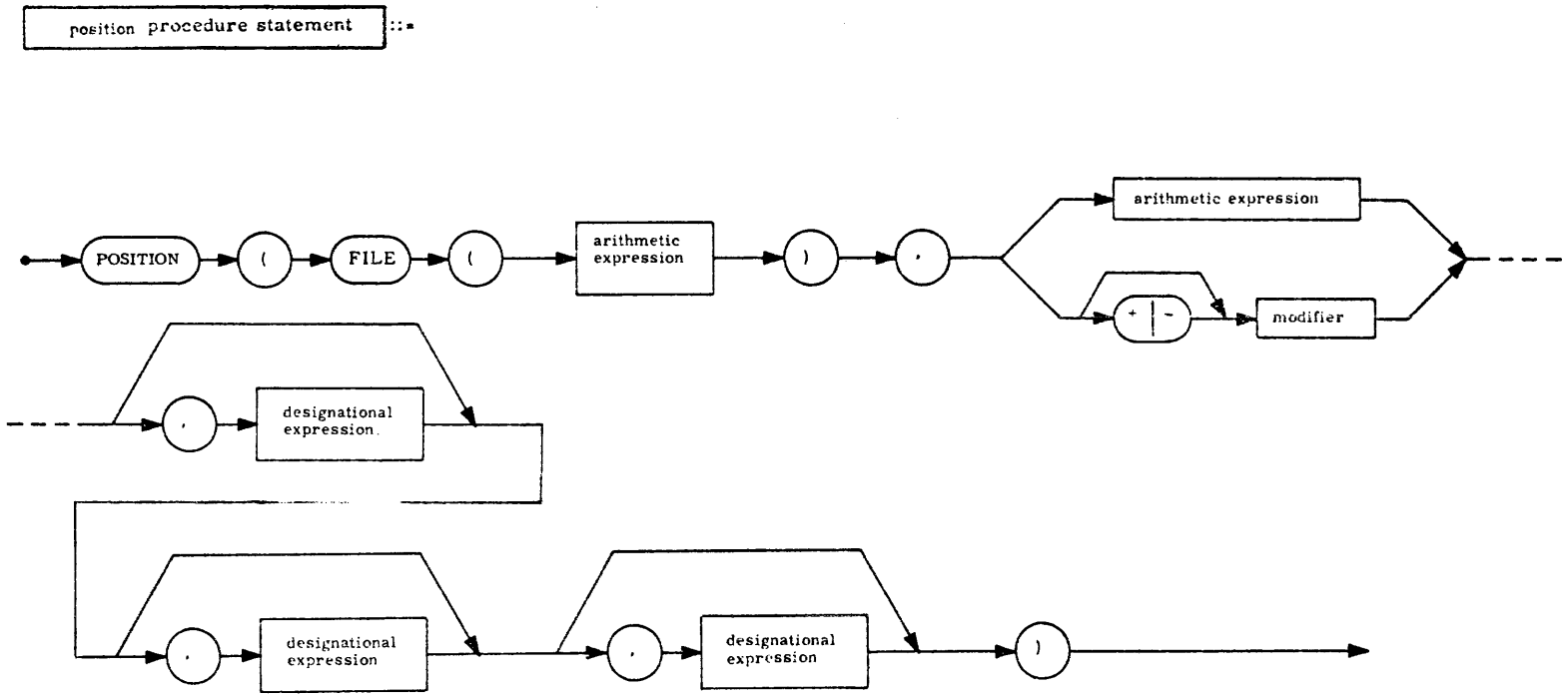
Examples:
```
READ(CARDS,LEOF,LEOI,A,B,C,S,EPSILON) $
READ(FILE(INDEX), FOR I=(1,1,KMAX) DO FOR J=(1,1,LMAX) DO ERG(I,J)) $
READ(DATE) $
```



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 point if the output device functions abnormally.

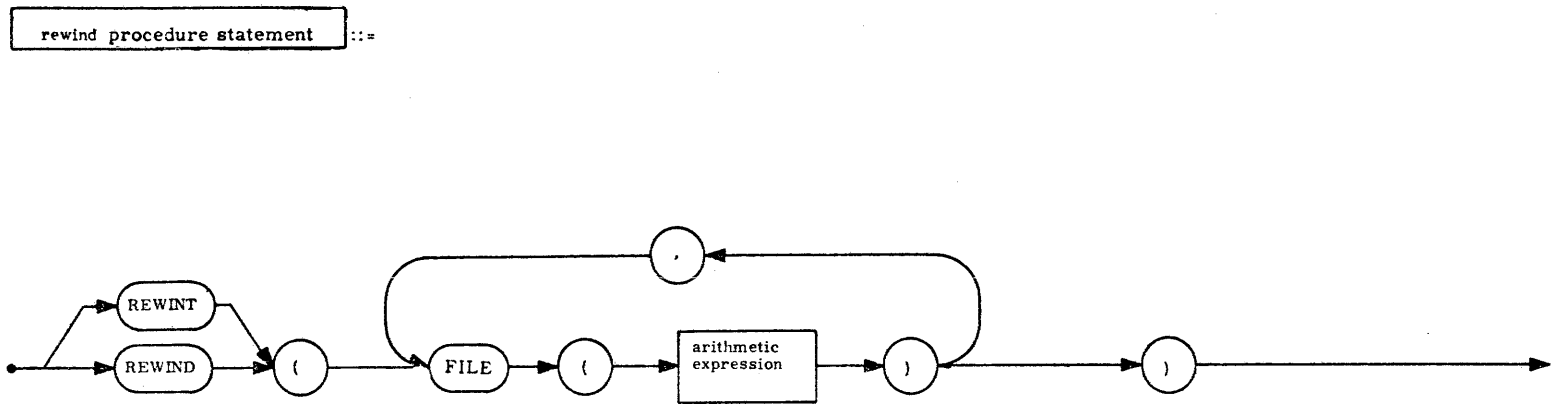
Examples:

```
WRITE (PRINTER, F10, FOR I=(1, 1, N) DO A(I,J)) $
WRITE ('CHECKPOINT CHARLIE', A) $
WRITE (FILE(0),KEY(I),ABORTLAB,DUMPLIST) $
WRITE (FILE(OUTPUT),EOF('LAST'),EOI) $
```



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 (FILE(0), -2) $
POSITION (FILE(INPUT), KEY('PRICES'), ABORT) $
POSITION (FILE(OUTPUT), EOI) $
```



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 (FILE (INPUT), FILE(OUTPUT)) $
REWINT (FILE(I),FILE(A),FILE(J)) $
```

APPENDIX F. EXEC II NU ALGOL

This appendix describes those operations of NU ALGOL under EXEC II (1107) that differ from corresponding operations under EXEC 8. Each item is preceded by a reference to this manual that describes operation under EXEC 8.

Reference	Difference or Addition for EXEC II
(2.2)	Identifiers
	d) One exception is for external procedures under EXEC II where only the first 6 characters are unique.
(2.5.2)	For EXEC II, one additional standard procedure CHAIN.
(3.1)	REAL2 On the 1107, the limits are the same for type REAL, but up to 16 digits for REAL2.
(4.2.3)	REAL2 Add 1107 having between 9 and 16 significant digits.
(7.3.3)	Assembler language on 1107 is called SLEUTH II; under EXEC II only six characters of the procedure name are marked as an entry point:

EXAMPLE:

1. externally compiled procedure

```
∇  E      ALG     < name >
```

main program

```
∇         ALG     < main name >
```

2. externally compiled procedure

```
∇  E      ALG     < name >
```

main program

```
∇         ALG     <main name >
```

(7.3.4) EXAMPLE:

```
∇ FOR          name 1
∇ ALG          name 2
```

Reference — Difference or Addition for EXEC II

(7.3.5) EXEC II - 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.

(7.3.5.1) PROGRAM EXAMPLES:

Assembler Language Program

```
 ▽ ASM          < name 1 >
```

```
instead of ER      ERR$
EXEC II    J       MERR$
```

Main Program

```
 ▽ ALG          < name 2 >
```

(7.3.5.2) Example for return from a LIBRARY procedure.

```
 ▽ ALG     MAIN     (EXEC II)
```

after END MAIN PROGRAM;

```
 ▽ ASM     PUNP
```

(7.4.1) Available Procedures

Name	Number of Parameters	Types of Parameters	Result
CHAIN EXEC II MARGIN	1	Integer	call link X in MAP
MARGIN EXEC II	3 or 4	Integer Integer Integer String	
TAPE	1	Integer String	Directing I/O from or to a specified sequential file.

Reference	Difference or Addition for EXEC II
(8.3.1)	Add device for EXEC II TAPE (see following information on EXEC II file handling)
(8.3.6.1)	Device for Sequential Files TAPE (<file name>) TAPE is also implemented under EXEC 8 to provide compatibility with EXEC II NU ALGOL Meaning of<file name> under EXEC II If the file name is an integer it must be in the range 0 to 20. If it is a string the first letter of the string must be one of the letters A through F. This letter is converted to integer so that A corresponds to 0, B to 1 etc. This letter is the same as the logical unit assigned to a magnetic tape. The integer file name is an index to an installation defined file control table called Y$TTAB. It is possible for the user to supply his own Y$TTAB table, 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 drum files 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		1st quarter	
10	Allowed	2nd quarter	
11		3rd quarter	
12		4th quarter	
13	↓	1st eight	

Reference

Difference or Addition for EXEC II

(8.3.6.1) (cont)

Y$TTAB (cont)

Parameter		Meaning
Integer	String	
	Tape simulating files	Drum layout
14	Allowed	2nd eight
15	↓	3rd eight
16		4th eight
17		5th eight
18		6th eight
19		7th eight
20		8th eight

(8.3.6.2) Under EXEC II device DRUM refers to the random access user file which consists of the user PCF and the processor scratch area. The size of this file will be installation dependent. To provide compatibility, DRUM may also be used under EXEC 8. In this case a temporary file of 20,000 words on word-addressable drum is automatically assigned.

Restriction Under EXEC II

Under EXEC II 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 user PCF area.

Speed Considerations

1. Parameters in a list are automatically placed in consecutive locations on the file.

   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.

2. 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$
   ```

Reference	Difference or Addition for EXEC II

(8.3.6.2) (cont)

3. 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$
```

(8.8.5) MARGIN under EXEC II has the form:

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

- <length> is an integer expression specifying the number of lines 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$
```

(9.2) The utility statements SET and RESET are not available in EXEC II NU ALGOL.

(9.3) OPTIONS - add for EXEC II:

F - On the 1107, this option must be used when using external FORTRAN, procedures containing double precision or complex arithmetic. Otherwise the program will terminate with the

Reference	Difference or Addition for EXEC II

(9.3) (cont)

message:

ILLEGAL OPERATION AT LINE XXX

where the line number refers to the last ALGOL line executed.

(9.4) Add 9.4

9.4 CHAINED PROGRAMS AND NU ALGOL (APPLIES ONLY UNDER EXEC II)

1. The EXEC II manual, 6.2, 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. The standard Y$$TAB occupies the first 150 words of blank common storage.

(10.2) Error Number 53

ERROR NUMBER	MESSAGE	POSSIBLE PROBLEM
53	Too many different identifiers	Approximately 600 different identifiers may be used on a 32 K EXEC II computer.

(Appendix D) D.1.1 Add

- On the 1107 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'.

INDEX

A

B

C

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

CUT

FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

BUSINESS REPLY MAIL NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

SPERRY UNIVAC

P.O. BOX 500
BLUE BELL, PA.
19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

FOLD

CUT