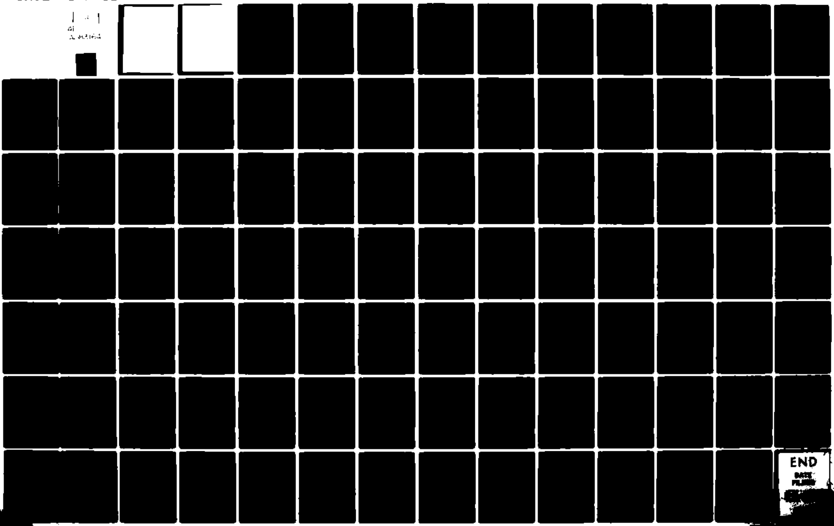
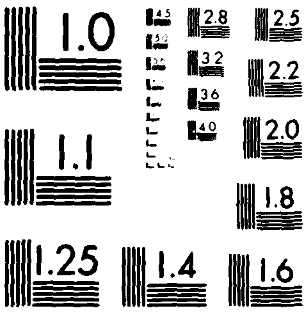
AD-A083 164 NAVAL SURFACE WEAPONS CENTER DAHLGREN VA F/G 9/2
TRIDENT HIGHER LEVEL LANGUAGE SYNTAX DEFINITION.(U)
NOV 79 P SHEBALIN
UNCLASSIFIED NSWC/TR-3914 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) NSWC/TR-[redacted] [illegible]	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) TRIDENT HIGHER LEVEL LANGUAGE SYNTAX DEFINITION.		5. TYPE OF REPORT & PERIOD COVERED Final [illegible]
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (11) Paul Shebalin		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center Code K51 Dahlgren, VA 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS OMN
11. CONTROLLING OFFICE NAME AND ADDRESS Strategic Systems Project Office Washington, DC 20390		(11) 12. REPORT DATE Nov[redacted]79
		13. NUMBER OF PAGES 96 (12) [illegible]
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

TRIDENT
language
programming
software
computer
fire control
syntax

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document presents a complete and rigorous syntactic definition of the TRIDENT Higher Level Language (THLL). Two different formulations are included: syntax graphs and productions. The syntax graphs provide a visual aid to quickly determine the structure of all THLL constructs. The productions express the syntax as it is used, essentially, by the compiler. → to p.

411567

FOREWORD

Con't

The TRIDENT Higher Level Language (THLL) is a procedure-oriented programming language for use in writing software programs for the TRIDENT Digital Control Computer (TDCC). This computer is to be used in the Mk 98 FCS and the Mk 88 Mod fire control system (FCS).

The purpose of this document is to provide a complete and rigorous syntactic definition of THLL.

This technical report supersedes NSWC/DL TN-K-9/78, dated May 1978.

Questions, comments, and suggestions regarding the material presented herein should be directed to the Fire Control Programming Branch, Support Software Group.

Released by:

[illegible]

R. T. RYLAND, JR., Head
Strategic Systems Department

Accession For	
NTIS GRA&I	✓
DDC TAB	
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A	

CONTENTS

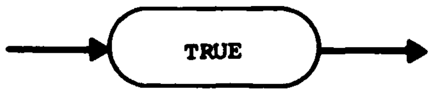
LIST OF TABLES

INTRODUCTION

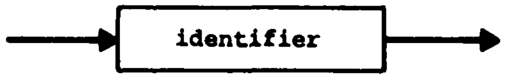
The purpose of this document is to provide a complete and rigorous syntactic definition of the TRIDENT Higher Level Language (THLL). Contractor and NSWC personnel who work with THLL are provided herein with a validation tool which will allow a quick determination of a program's syntactic correctness. The TRIDENT Higher Level Language User's Guide (NSWC TR-3657 Revised June 1978) should be consulted for any questions concerning the semantics or pragmatics of THLL programming.

The grammatical definition of THLL given here consists of a set of syntax graphs and a Backus-Naur Form (BNF) description (Appendix A).

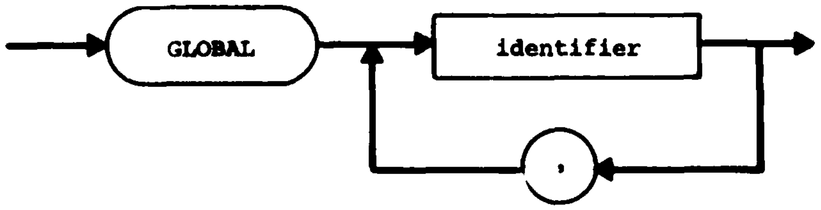
In the syntax graphs, a distinction is made between terminal and non-terminal elements by representing a terminal with capital letters or special characters and enclosing it in a smooth, closed curve:



A non-terminal, corresponding to the left-hand side of a BNF production, is represented with lower case letters enclosed by a rectangle:



Correct sequences of terminals and non-terminals, again corresponding to the right-hand side of a BNF production, are interconnected with arrows:

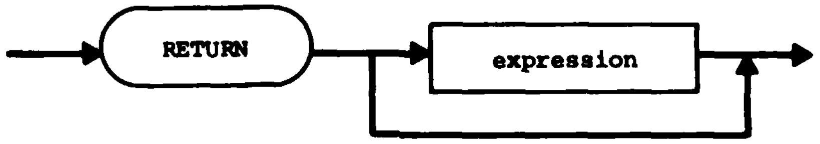


The BNF description of THLL, given in Appendix A, is essentially a bottom-up presentation of the syntax. Generally, each syntax graph corresponds to a set of BNF productions. A BNF production consists of a left-hand side and a right-hand side separated by the symbol '!!='. The BNF's left-hand side is always a non-terminal of the grammar, while the right-hand side consists of a set of alternations, each alternation consisting of a sequence of grammar symbols. Non-terminals are represented with lower case letters enclosed by the symbols '<' and '>', and terminals are represented by capital letters or special characters. Alternations in a BNF production are separated by the backslash character, '\', for example:

```
<return statement> !!= RETURN \ RETURN <expression>
```

which corresponds to the syntax graph

<u>return statement</u>



Because a particular syntax graph may correspond to several BNF productions, the grammar defined by the syntax graphs has fewer non-terminal elements than the BNF grammar. Thus, the syntax graph representation of the THLL grammar is somewhat coarser than the BNF representation. For example, the precedence of arithmetic and logical operators is not reflected in the syntax graphs.

Much of the explanatory text for this report was taken from the THLL User's Guide (NSWC TR-3657).

SYNTAX GRAPHS

BASIC SYMBOLS, CONSTANTS AND IDENTIFIERS

On the lowest level, a THLL program is a character string. Characters are grouped together as items which fall into one of the following categories:

1. Operators
2. Delimiters
3. Constants
4. Symbols (identifiers)

There is a fixed number of operators and delimiters and these are listed in Tables 2 and 3.

The character set includes the English letter alphabet (A-Z), the numerals (0-9), a single space, and certain special characters (see Table 1).

Table 1. Character Set

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z	0	1	2	3	4	5	6	7
8	9	-	+	*	/	(	)	=	,	.	:	;	#	"	$	!
'	<	>	%	_		&	?	@	{ / [	\	} /]	∧ / ±				

Note: Where two characters appear, the upper one is used for coding purposes to cause the lower one to appear on the TRIDENT Digital Control Computer (TDCC) output devices.

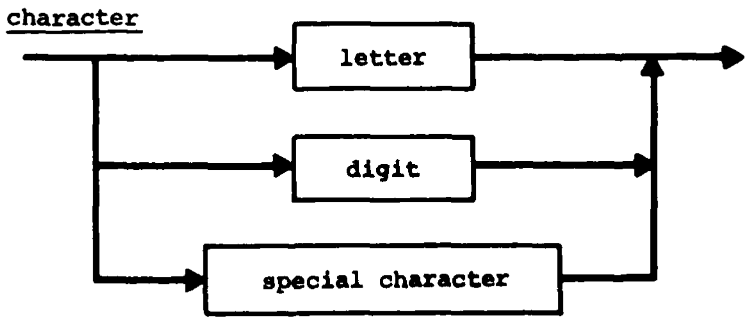


Table 2. THLL Operators

Class	Mnemonic	Meaning
Arithmetic	+	addition
Arithmetic	-	subtraction
Arithmetic	*	multiplication
Arithmetic	/	division
Arithmetic	**	exponentiation
Arithmetic	MOD	modulo
Relational	LES	less than
Relational	LEQ	less than or equal
Relational	EQL	equal
Relational	GRT	greater than
Relational	GEQ	greater than or equal
Relational	NEQ	not equal
Logical	OR	or
Logical	XOR	exclusive or
Logical	AND	and
Logical	NOT	not
Assignment	=	assignment of value
Bit	ANDB, BITAND	and bits
Bit	ORB, BITOR	or bits
Bit	XORB, BITXOR	exclusive or of bits
Bit	NOTB, BITNOT	not bits
Addressing	LOC	LOC X is the virtual[1] address of the word containing the first bit of X.
Addressing	LOCA	LOCA X is the absolute address of the word containing the first bit of X.
Addressing	ENTRYP	ENTRYP X is the virtual[1] address of the beginning of procedure X and is used to pass the procedure as a formal parameter.

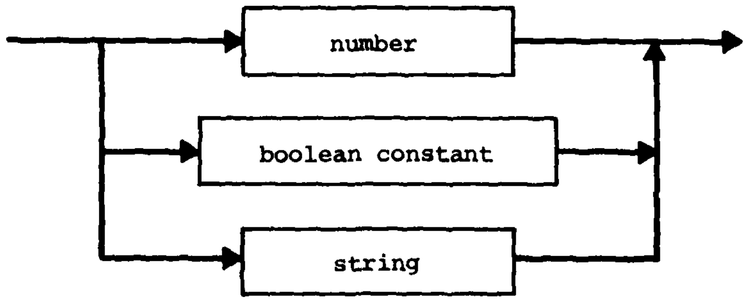
[1] Absolute address on the CDC 6700.

Table 3. Delimiters

ALPHA	FIELD	$
ARITHMETIC	FINIS	POINTER
ARRAY	FOR	PRESET
BEGIN	FORMAT	PROCEDURE
CASE	GLOBAL	REAL
CASEEND	GOTO	REPEAT
COMEND	HALF	RETURN
COMMENT	ICL	SPRINT
COMMON	IF	STACK
COMPONENT	IFEND	STEP
CPRINT	INSERT	SWITCH
DEFINE	INTEGER	SYNONYM
DEVICE	INTERRUPT	TASS
DO	KBDSS	THEN
DOUBLE	LINK	TO
ELSE	LOGICAL	UNTIL
END	LOOPEXIT	VALUE
ENDCASE	MDF	WHILE
ENDCOM	MTF	;
ENDIF	NULL	,
EXEC	OFFSET	:
EXIT	OPTARG	(
EXTERNAL	OWN	)

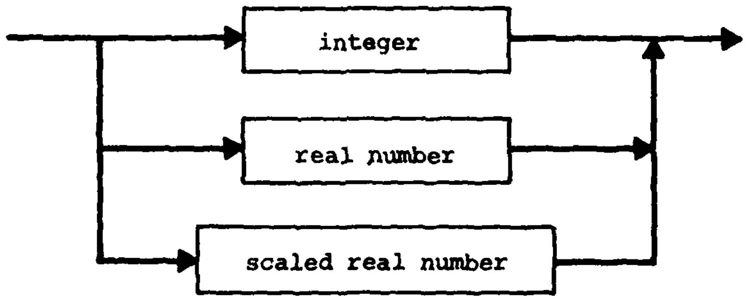
Constants

constant



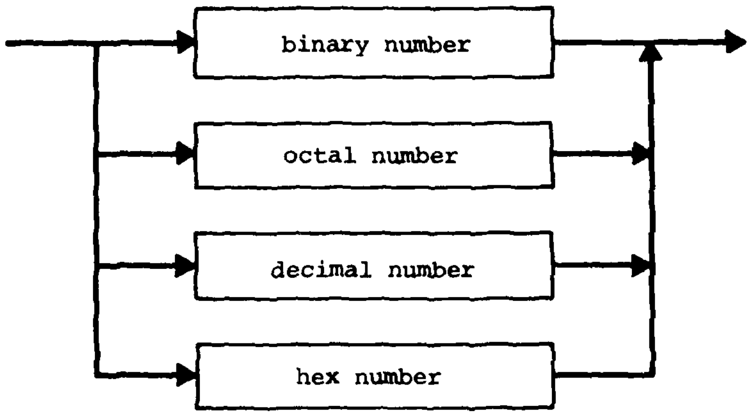
Numbers

number

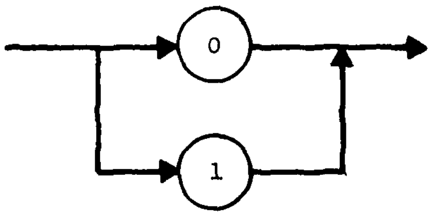


Integers

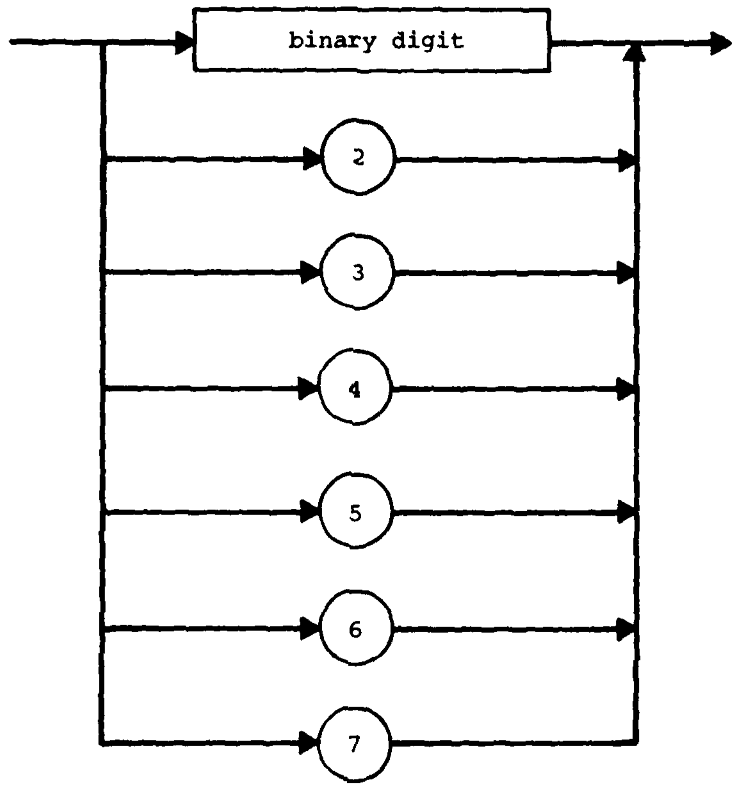
integer



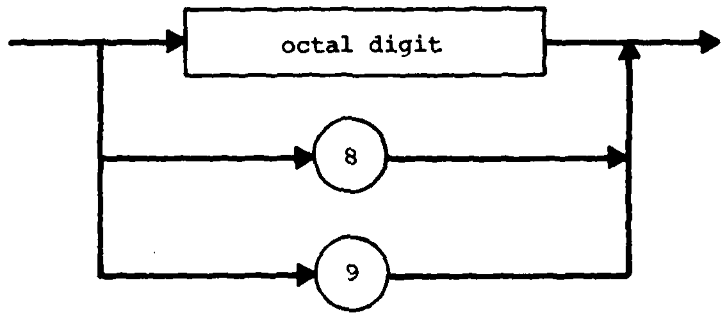
binary digit



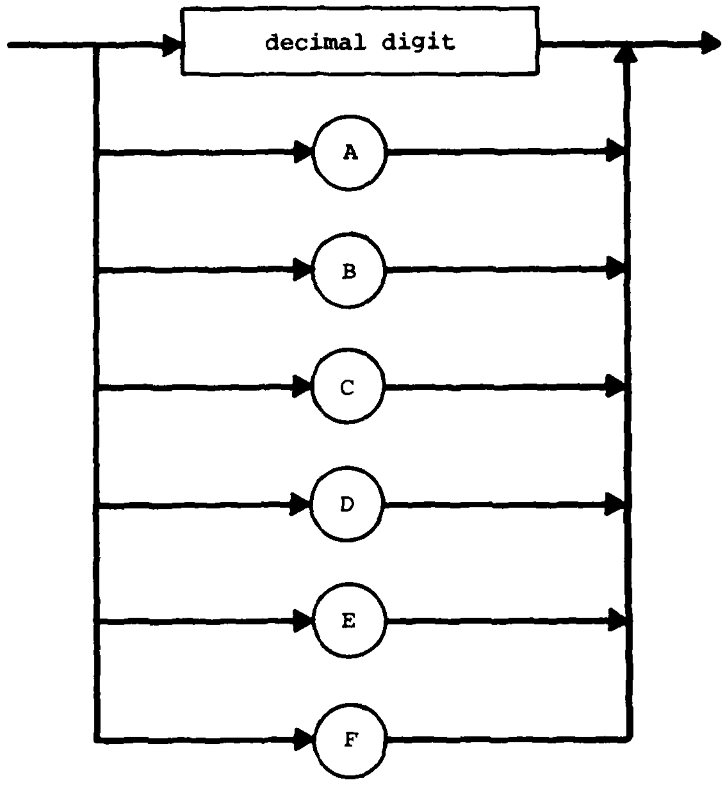
octal digit



decimal digit

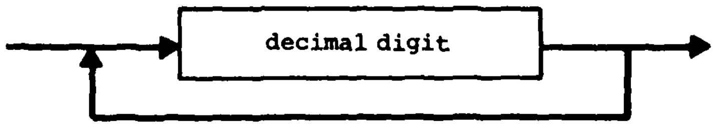


hex digit



A decimal integer is simply a sequence of decimal digits.

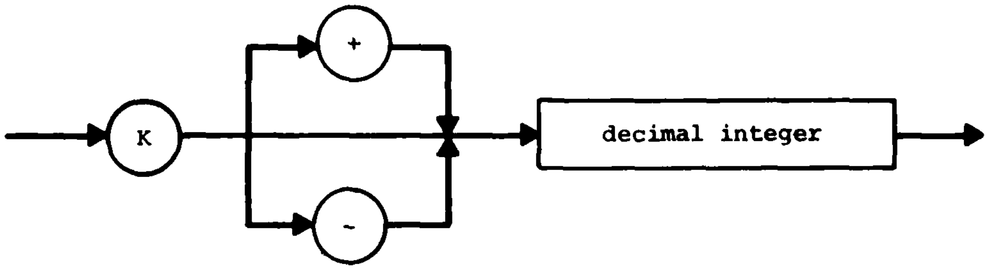
decimal integer



Similarly, binary, octal, and hexadecimal integers are sequences of binary, octal, and hexadecimal digits, respectively.

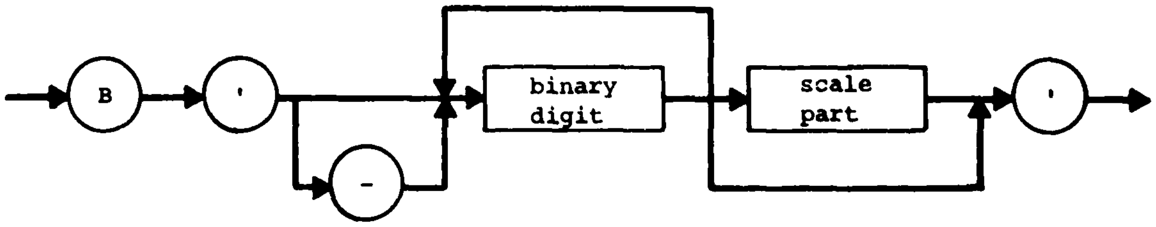
A scale part may be appended to a binary, octal, or hexadecimal integer to partially form a binary, octal, or hexadecimal integer, respectively. The scale part specifies a power of 2.

<u>scale part</u>

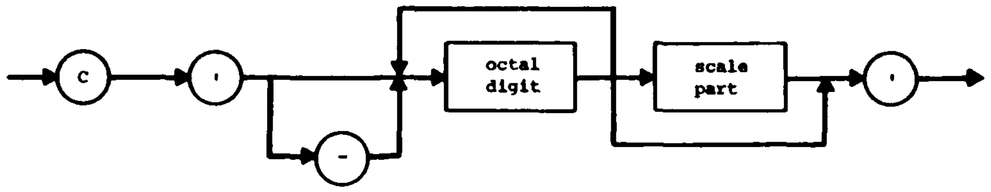


To complete the representation of a binary, octal, or hexadecimal number, the integer part and optional scale part may be preceded by a minus sign; and the resulting string is surrounded by quotation marks and prefixed with the appropriate character.

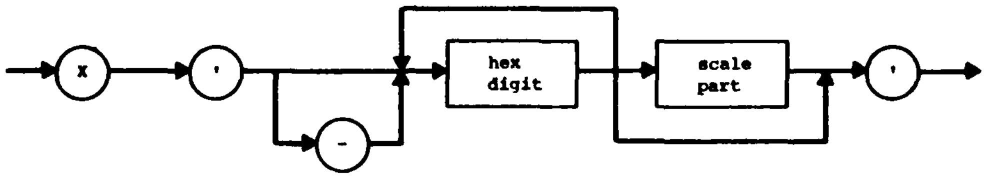
<u>binary number</u>



octal number

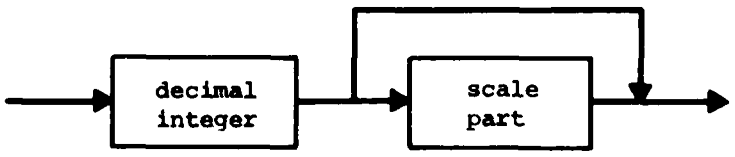


hex number



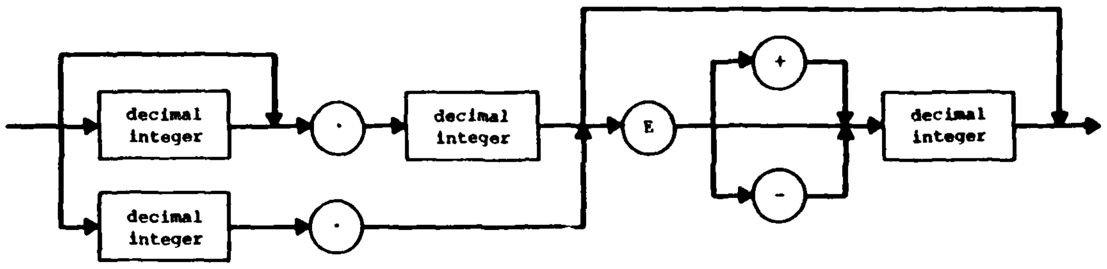
A decimal number consists only of a decimal integer followed by a scale part.

decimal number



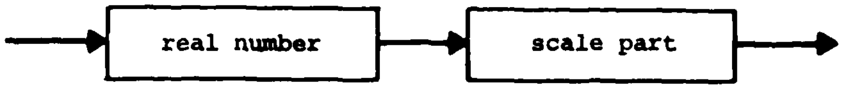
Real Numbers. A real number consists of a decimal floating point number followed by an optional, base 10 exponent.

real number



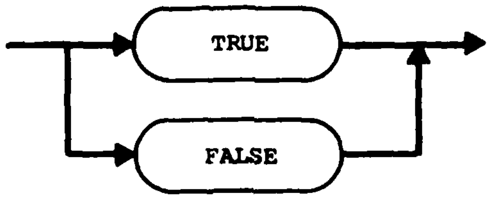
Scaled Real Numbers

scaled real numbers



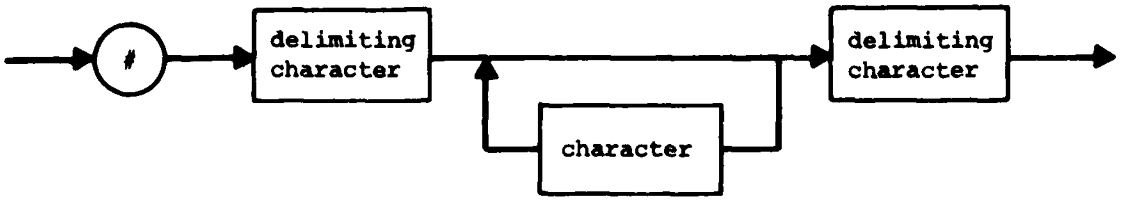
Boolean Constants. In THLL, the logical values for 'true' and 'false' are represented by the strings, TRUE and FALSE.

boolean constant



Strings. A literal character string is denoted by a '#' followed by a character sequence flanked by identical delimiting characters. The delimiting character can be any of the THLL characters as long as it is not contained in the character sequence comprising the string. The string length may be from 0 to 256 characters.

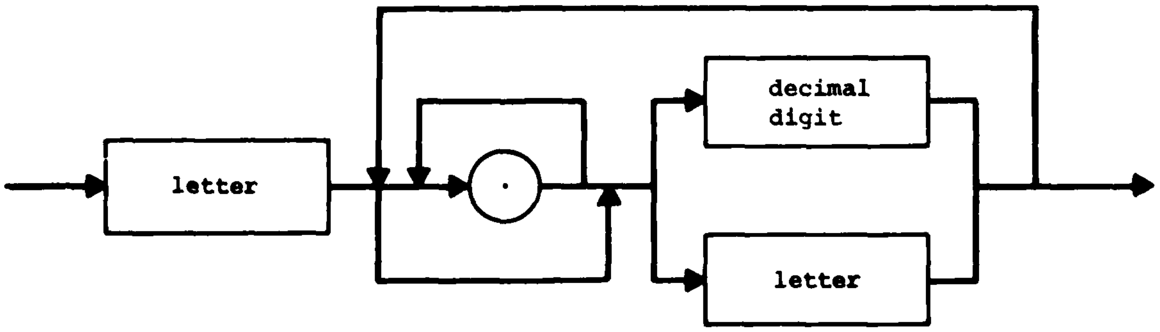
string



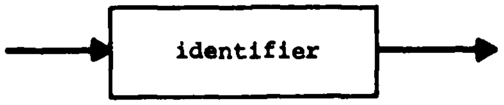
Identifiers

A THLL identifier is a user-defined name which can denote a component, array, stack, device, procedure, format, variable, label, or switch. Although an identifier may be represented by up to 256 characters, only the first eight determine its uniqueness. 'Letter' may be any of the 26 alphabetic characters.

identifier



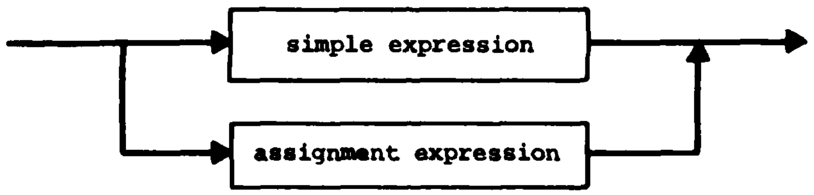
component id, array id, stack id, device id, procedure id, format id, variable id, label id, switch id



EXPRESSIONS

An expression is a rule for computing a new value from existing values. Expressions are built from constants, variables, function designators, and operators.

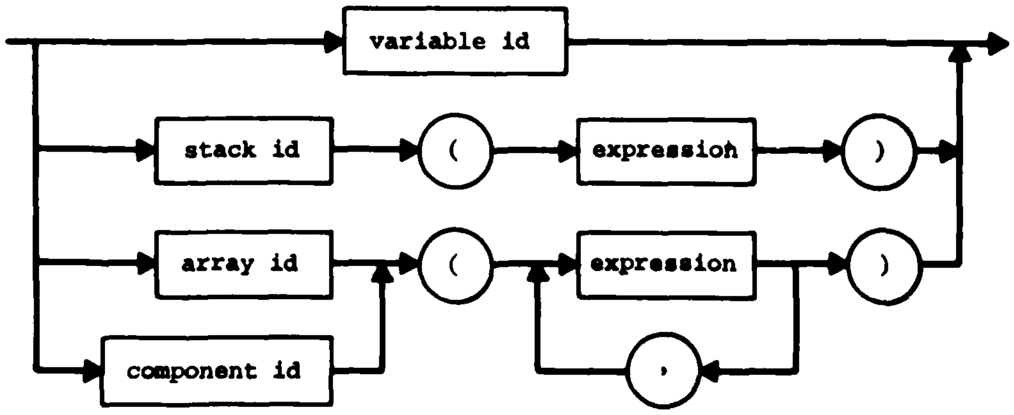
expression



Variables

Variables may either be simple or subscripted. An array may have a maximum of three subscripts and a component may have a maximum of two.

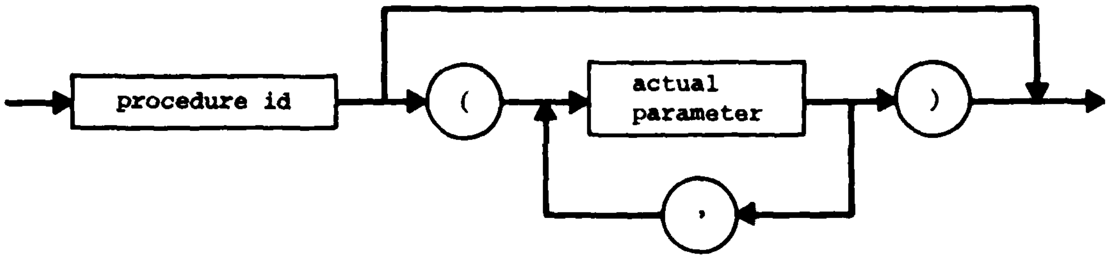
variable



Function Designators

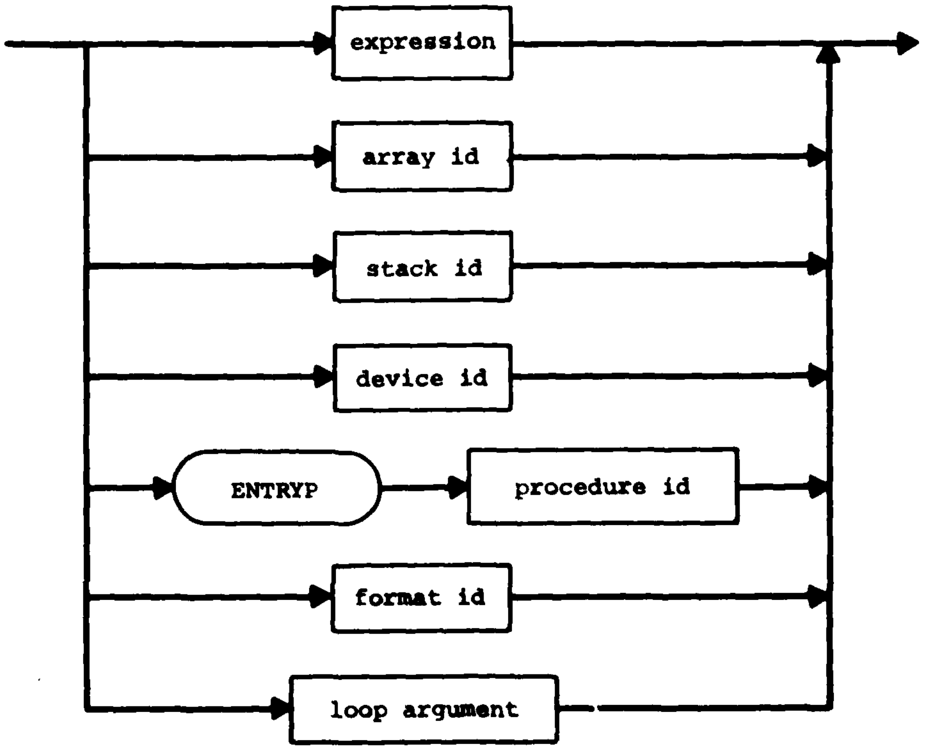
A function designator is the application of a procedure to a fixed set of parameters, resulting in a value. It is an expression and must be of type I, D, R, or P.

function designator



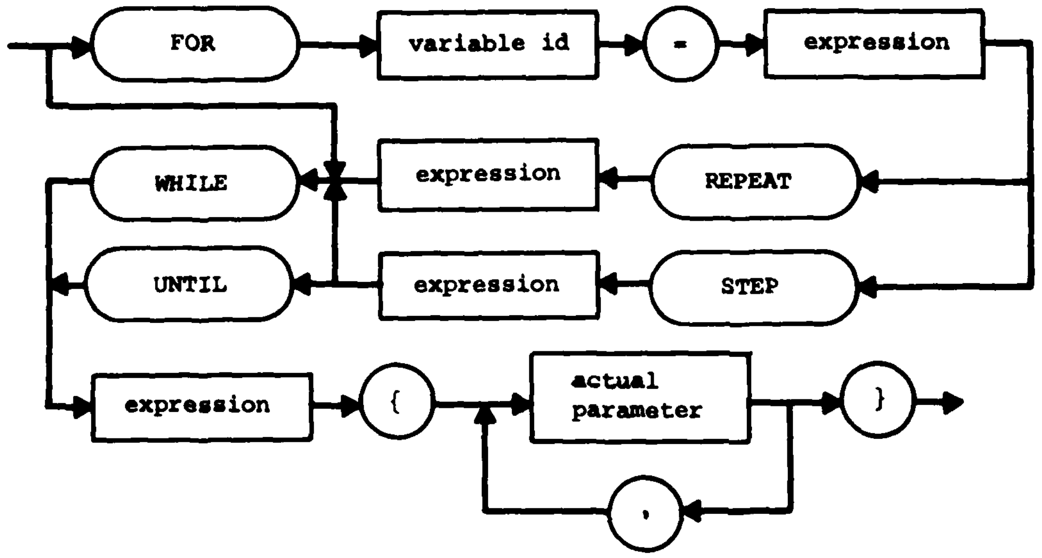
There is a set of predefined (standard) functions for which the user does not have to supply a declaration. These functions are listed in Tables 4 and 5.

actual parameter



Although it is syntactically correct to use a loop argument anywhere in a function parameter list, it is semantically correct to use a loop argument only after the fifth parameter in a READ/WRITE statement.

loop argument



In Tables 4 and 5, types of values are given using the following abbreviations:

I	integer
H	half
D	double
R	real
P	pointer
N	no type

Table 4. Numerical Functions

Function Name	Type of Argument(s)	Type of Value
ABS	(HIDR)	I, D, R correspondingly
SIGN	(HID)	I, D (1 or -1)
	(R)	R (1. or -1.)
SQRT	(HIDR)	R
SIN	(HIDR)	R
COS	(HIDR)	R
TAN	(HIDR)	R
COT	(HIDR)	R
ARCCOS	(HIDR)	R
ARCSIN	(HIDR)	R
ARCTAN	(HIDR)	R
ARCCOT	(HIDR)	R
LN	(HIDR)	R
EXP	(HIDR)	R
FLOAT	(HID)	R
FLOAT	$\left(\left\{\begin{matrix}R\\I\end{matrix}\right\}[,I]\right)$	R
FIXH	(R)	I
FIXI	$\left(\left\{\begin{matrix}R\\I\end{matrix}\right\}[,I]\right)$	I
FIXD	(R)	D
SHIFTA	$\left(\left\{\begin{matrix}H\\I\\D\end{matrix}\right\}, \left\{\begin{matrix}H\\I\\D\end{matrix}\right\}\right)$	I, D corresponding to argument 1

Table 4. Numerical Functions (Cont'd)

Function Name	Type of Argument(s)	Type of Value
SHIFTL	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}\right)$	I, D corresponding to argument 1
SHIFTR	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}\right)$	I, D corresponding to argument 1
TEST.BIT	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, P\right)$	I
CLR.BIT	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, P\right)$	I
SET.BIT	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, P\right)$	I
TGL.BIT	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, P\right)$	I
FIND.BIT	$\left(\begin{Bmatrix}H\\I\\D\end{Bmatrix}, P\right)$	I
POCA	(R,R,R,R)	N
CAPO	(R,R,R,R)	N
ROAX	(R,R,R,R,R)	N
ROTA	(R,R,R,R,R)	N

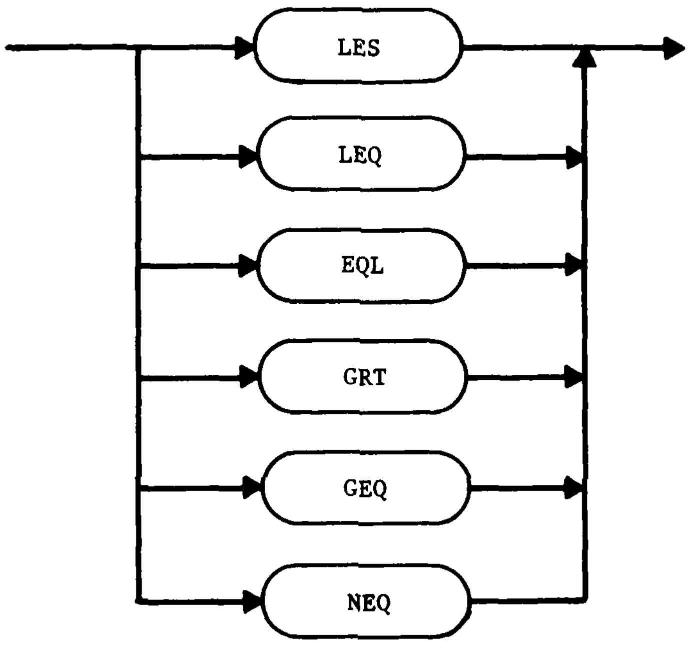
Table 5. Miscellaneous Functions

Function Name	Type of Argument	Type of Value
LENGTH	$\left(\begin{Bmatrix}A\\P\end{Bmatrix}\right)$	I
MLENGTH	$\left(\begin{Bmatrix}A\\P\end{Bmatrix}\right)$	I
MOVEC	$\left(\begin{Bmatrix}A\\P\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}A\\P\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}\right)$	N
CONC	$\left(\begin{Bmatrix}A\\P\end{Bmatrix}, \begin{Bmatrix}A\\P\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}A\\P\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}, \begin{Bmatrix}H\\I\\D\end{Bmatrix}\right)$	N
ORDERC	$\left(\begin{Bmatrix}A\\P\end{Bmatrix}, \begin{Bmatrix}A\\P\end{Bmatrix}\right)$	I
PUSH	(stack id, expression)	N
POP	(stack id)	N
STACKWC	(stack id)	I
STACKSC	(stack id)	I
BOUND	(array id, I)	I
SWA	(I)	N

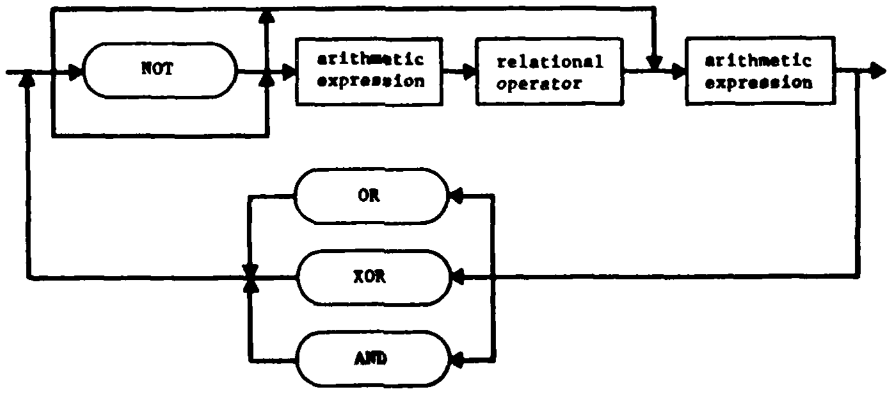
Simple Expression

Constants, variables, function designators, conditional expressions, case expressions, and location expressions are basic elements in THLL. Simple expressions are basic elements or constructed from these using the bit, arithmetic, relation, and logical operations.

relational operator

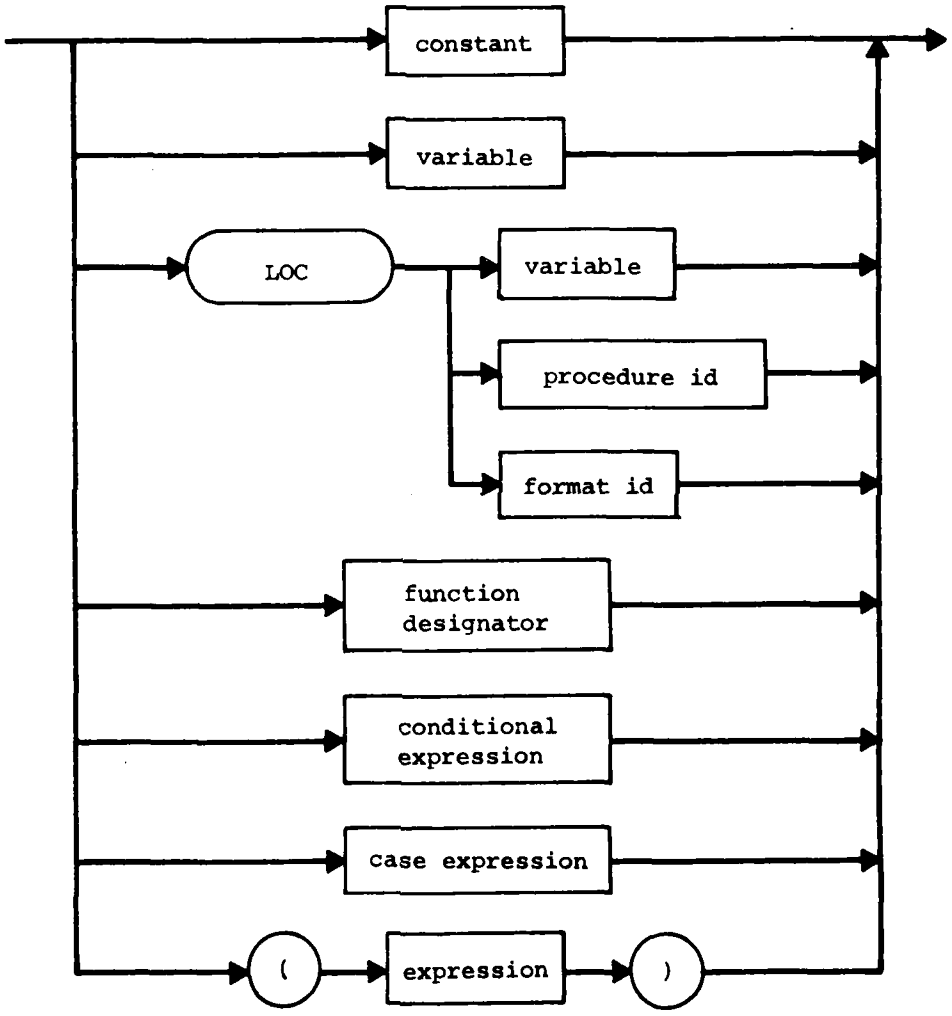


simple expression

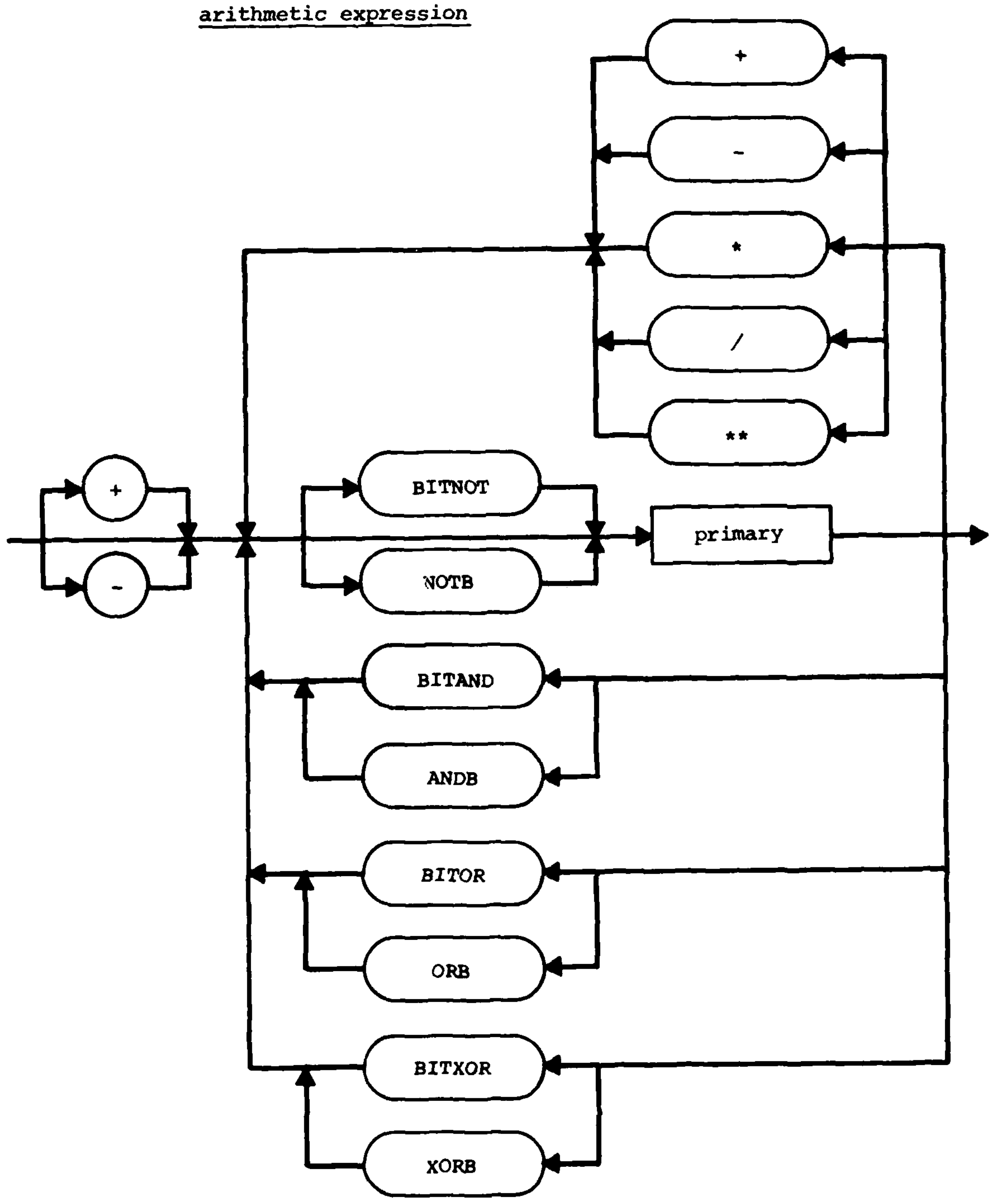


Arithmetic Expression. An arithmetic expression is a primary or basic element or a combination of basic elements connected by arithmetic or bit operators.

primary

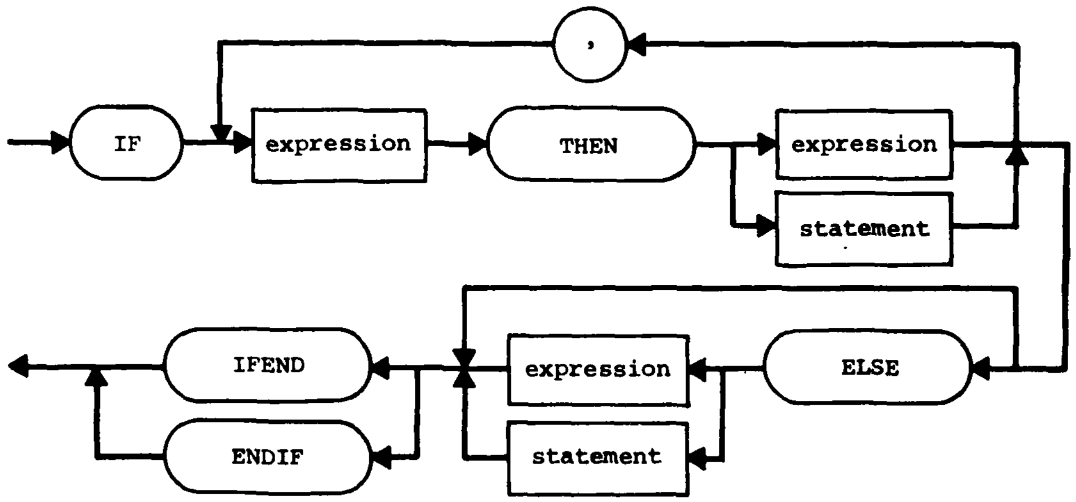


arithmetic expression



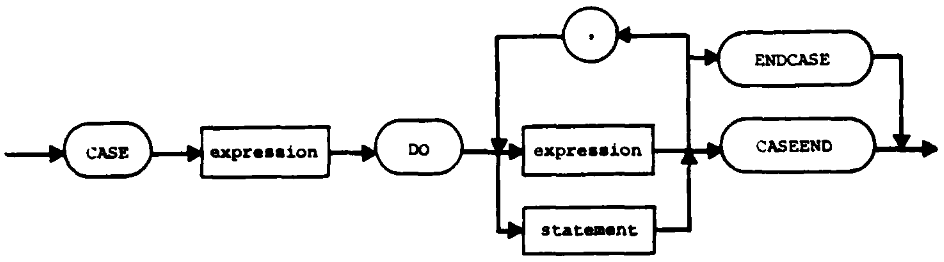
Conditional Expression. A conditional expression allows the evaluation of a statement or expression to proceed only if programmer-defined conditions have been met. The conditional expression becomes a conditional statement if the common type of all THEN expressions and the ELSE expression is of type statement.

conditional expression



Case Expression. The general form of the case expression, CASE P DO e_1, e_2,..., e_n ENDCASE, allows the evaluation of one of a set of indexed expressions. The expression, P, is evaluated to an integer which serves as an index to select which expression e_i will be done. If P produces a value less than 1 or greater than n, then P is assigned the value n. The case expression becomes a case statement if the common type of all DO expressions is of type statement.

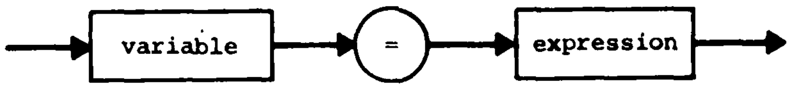
case expression



Assignment Expression

The value of an assignment expression is the value of the right-side expression. The assignment of this value to the left-side variable can be considered a side effect.

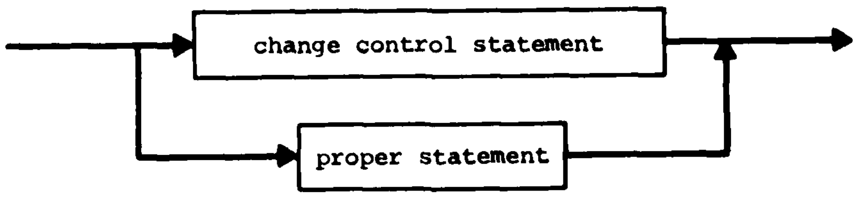
assignment expression



STATEMENTS

A THLL statement is a program unit which produces an effect on the environment.

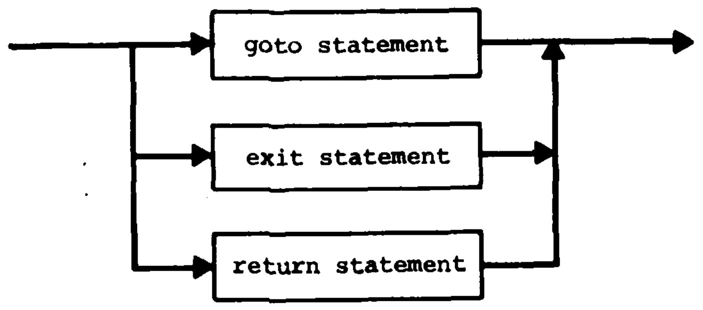
statement



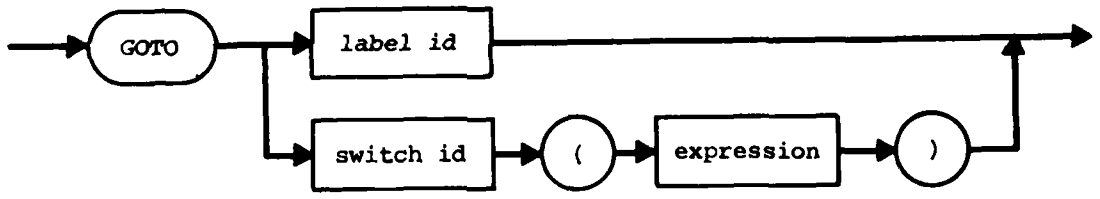
Change of Control Statement

Change of control statements transfer control to a designated point in the program or return control from a called procedure to the point of call.

change control statement

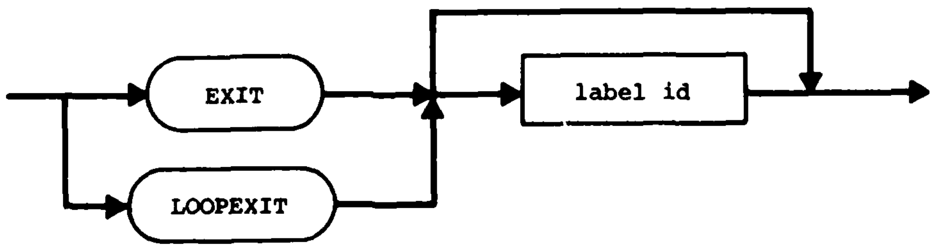


goto statement



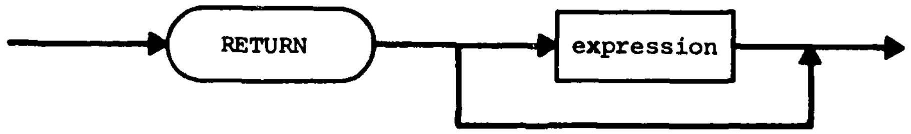
The effect of an exit statement is to transfer control to the end of the present block or to the end of an embracing block labeled by the indicated label identifier.

exit statement



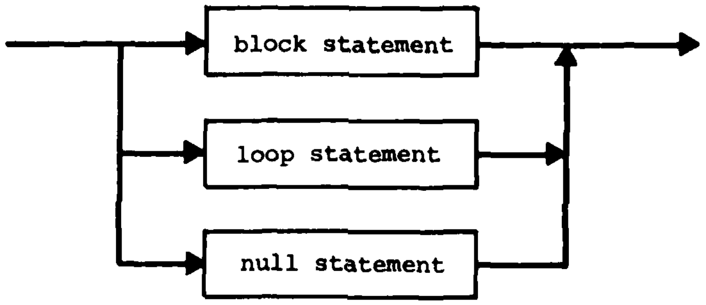
The effect of a return statement is to terminate the evaluation of a procedure body and transfer control to the calling procedure.

return statement



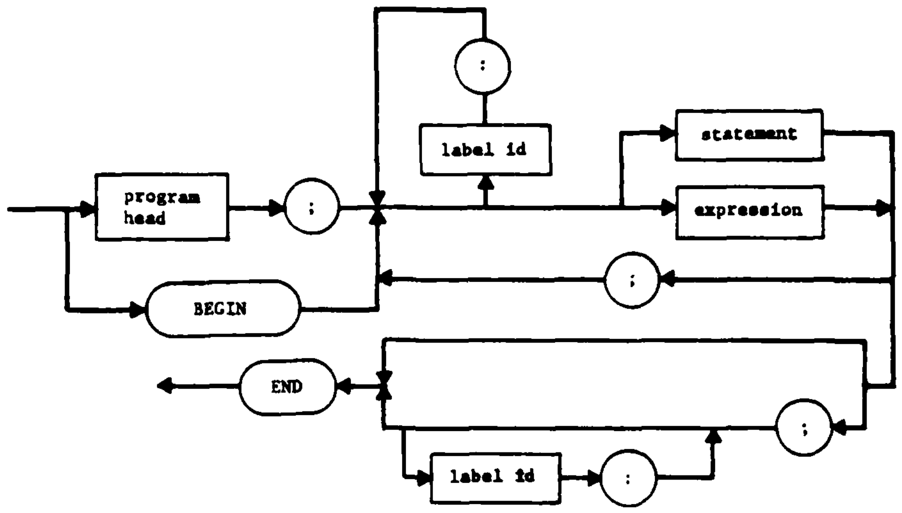
Proper Statements

proper statement



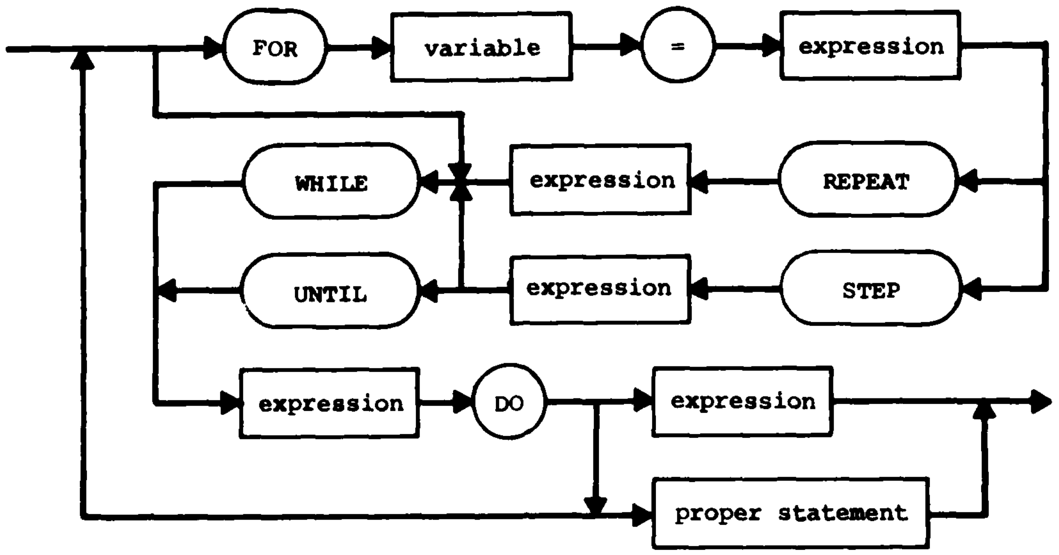
A block may be defined as a sequence of zero or more declarations followed by one or more statements or expressions, all separated by semicolons and embraced by BEGIN-END brackets.

block statement

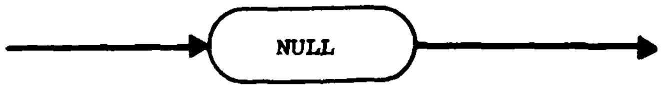


There are four types of loop statements. Each type used the reserved word DO followed by an expression or a proper statement, which will be evaluated zero or more times according to conditions.

loop statement



null statement

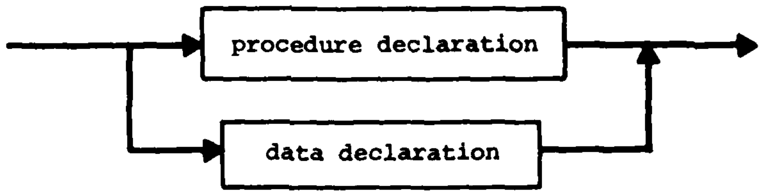


DECLARATIONS

The primary purpose of declarations is to provide information to the compiler about symbols used in the program. This includes data types, array and stack sizes, I/O formats, procedures, etc.

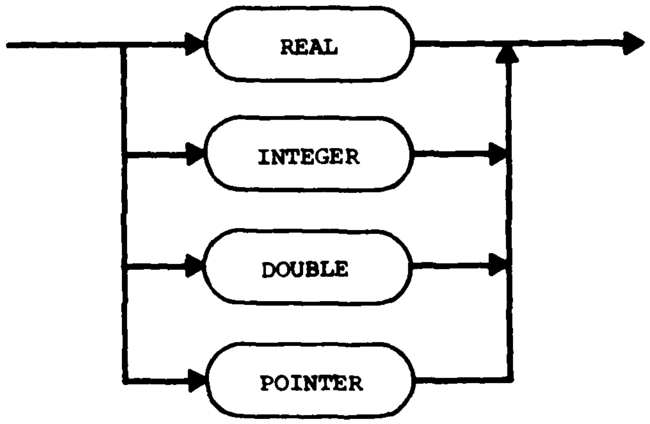
There are two categories: data and procedure. Every identifier which does not appear as a label must be declared. A procedure declaration defining an identifier to be a procedure name binds it to the procedure definition. All other declarations fall into the data category.

declaration

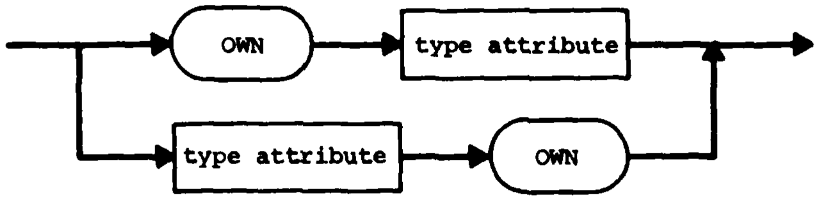


Attributes. Both data and procedures are given various attributes upon declaration. These attributes define type, size, and allocation mode in the case of data, and access and type in the case of procedures.

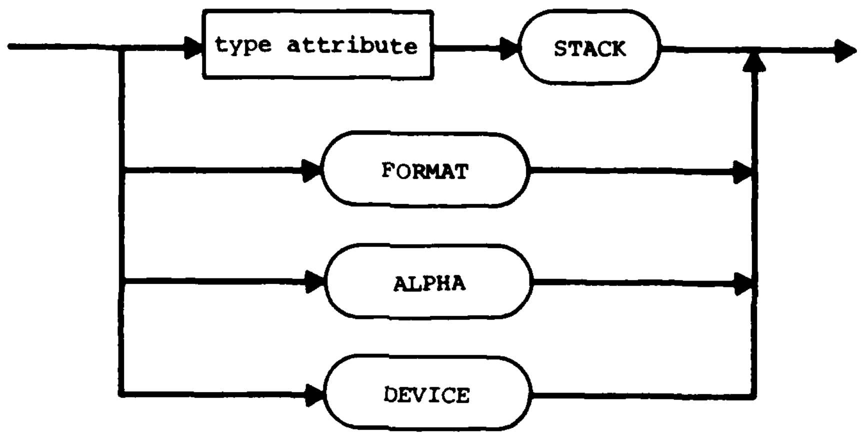
type attribute



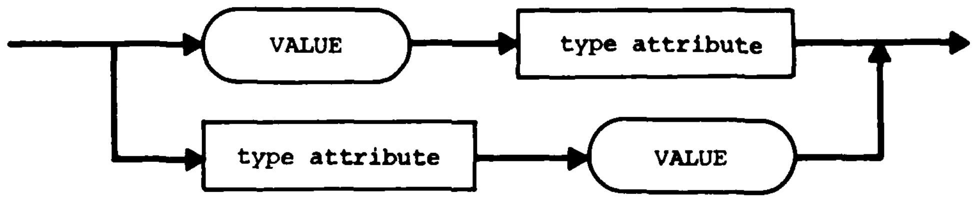
full attribute



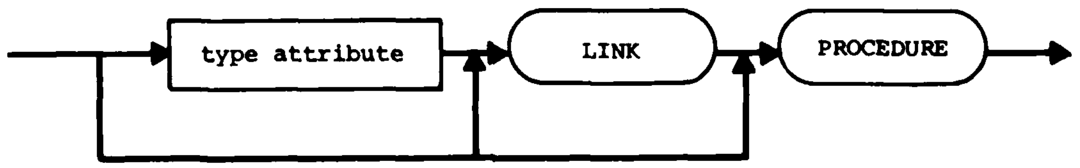
attribute head



value attribute

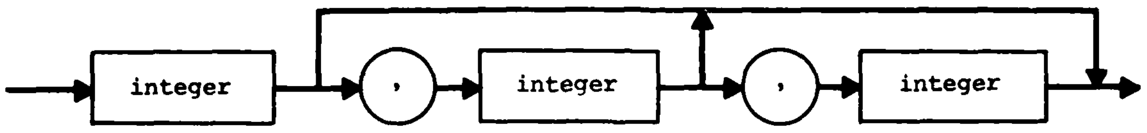


procedure attribute



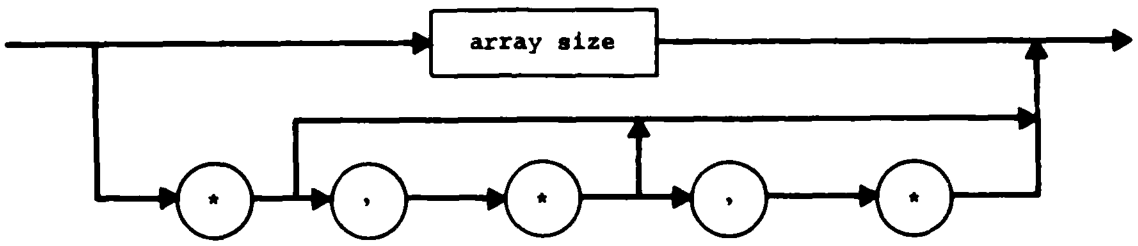
Arrays can be defined as having one, two, or three dimensions.

array size



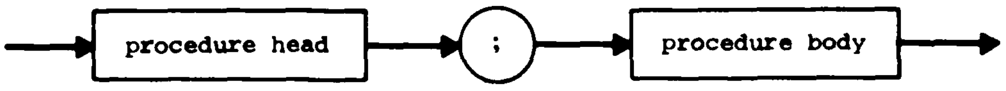
The lower bound for any dimension is always 0, while the upper bound is defined through an array declaration. An externally defined array can be declared with either its proper numerical dimensional bounds or with asterisks.

external size

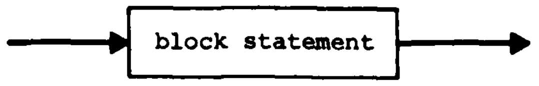


Procedure Declaration. A procedure declaration defines an identifier to be the name of a procedure. The main part of a procedure is the procedure body which is always a block that specifies the piece of code to be executed when the procedure is invoked.

procedure declaration



procedure body

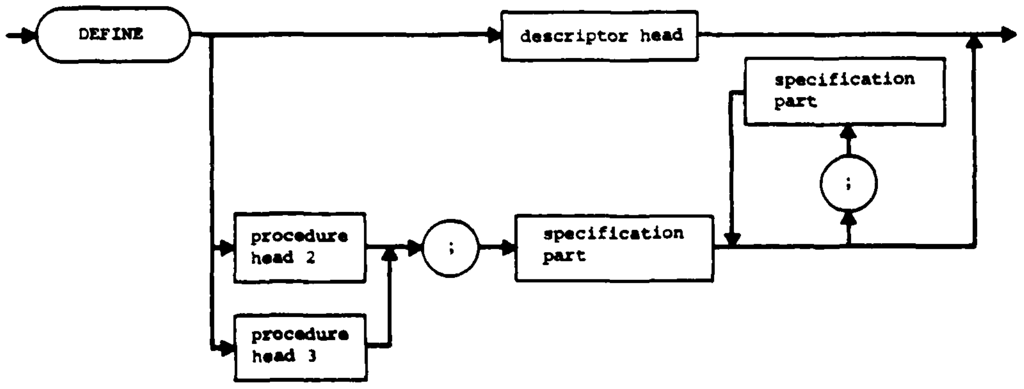


The procedure body is preceded by the procedure head which contains:

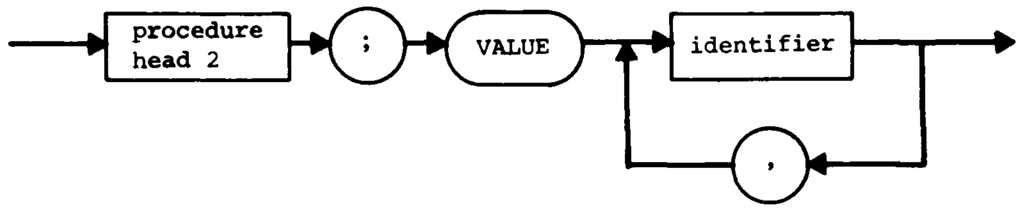
1. Access of the procedure
2. Type of the value of the procedure
3. Name of the procedure
4. List of formal parameters
5. Description of formal parameters

Items 1, 2, 4, and 5 may be omitted.

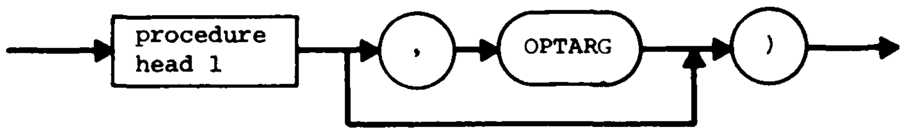
procedure head



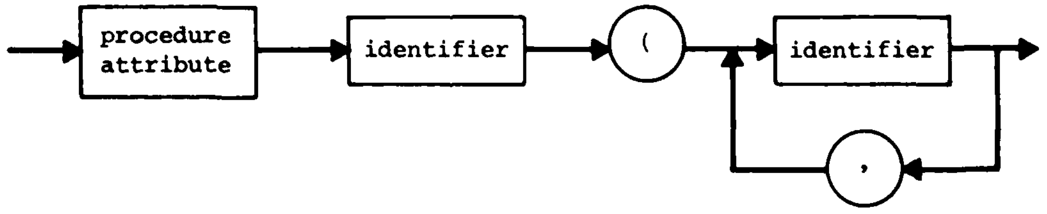
procedure head 3



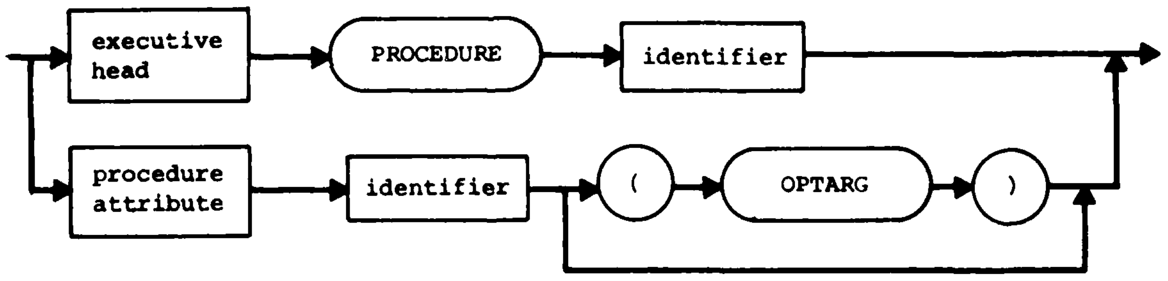
procedure head 2



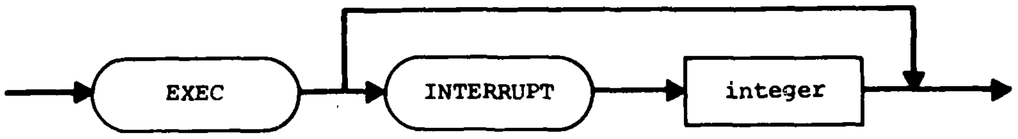
procedure head 1



descriptor head

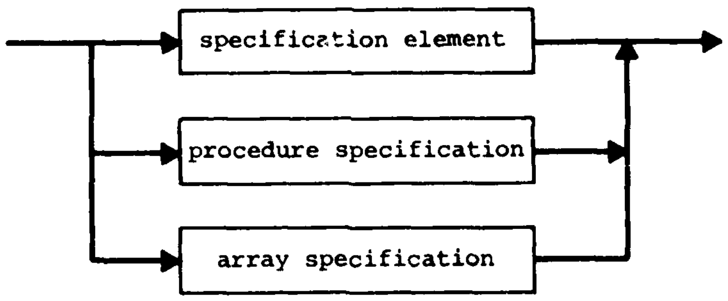


executive head

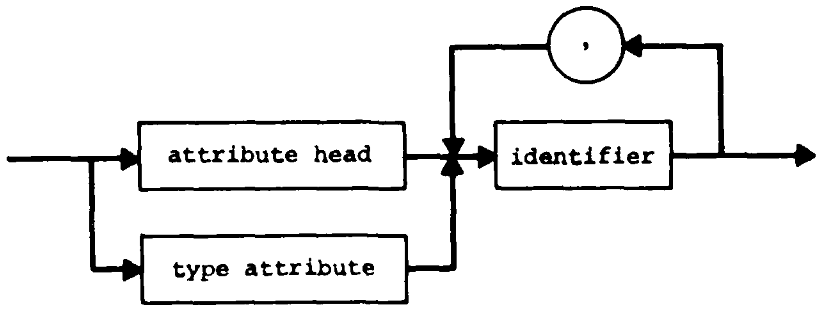


Every formal parameter must appear in the specification part. Those parameters which are to be transmitted by value will appear in the value part and the specification part. The specification part will follow the value part, if there is one, or the procedure head, if there is no value part.

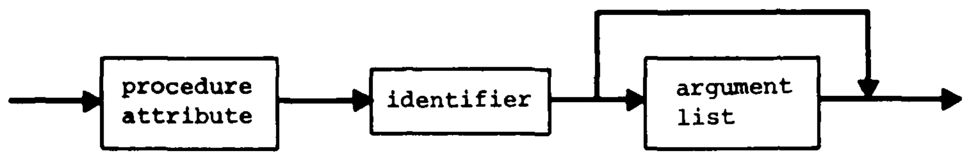
specification part



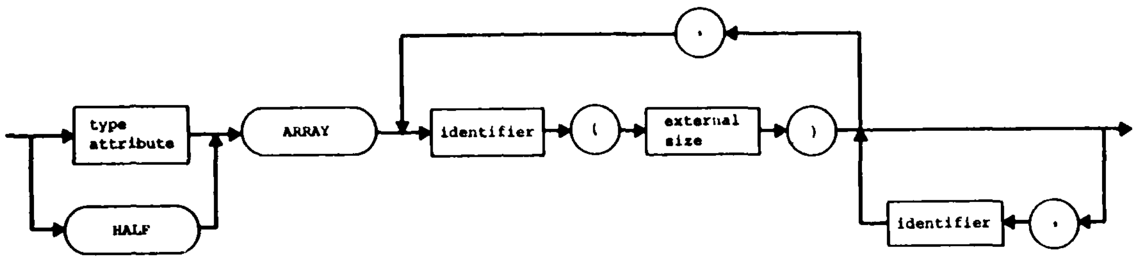
specification element



procedure specification

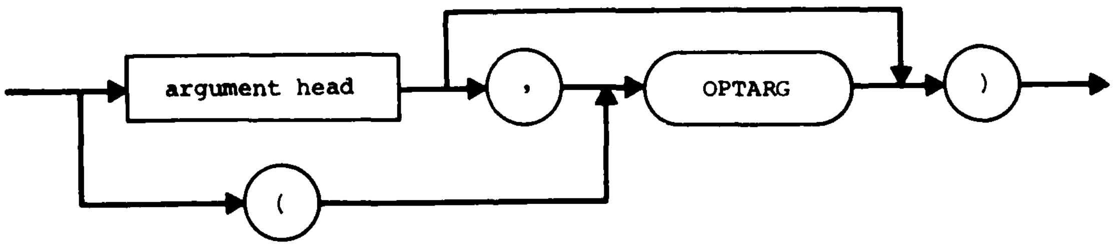


array specification

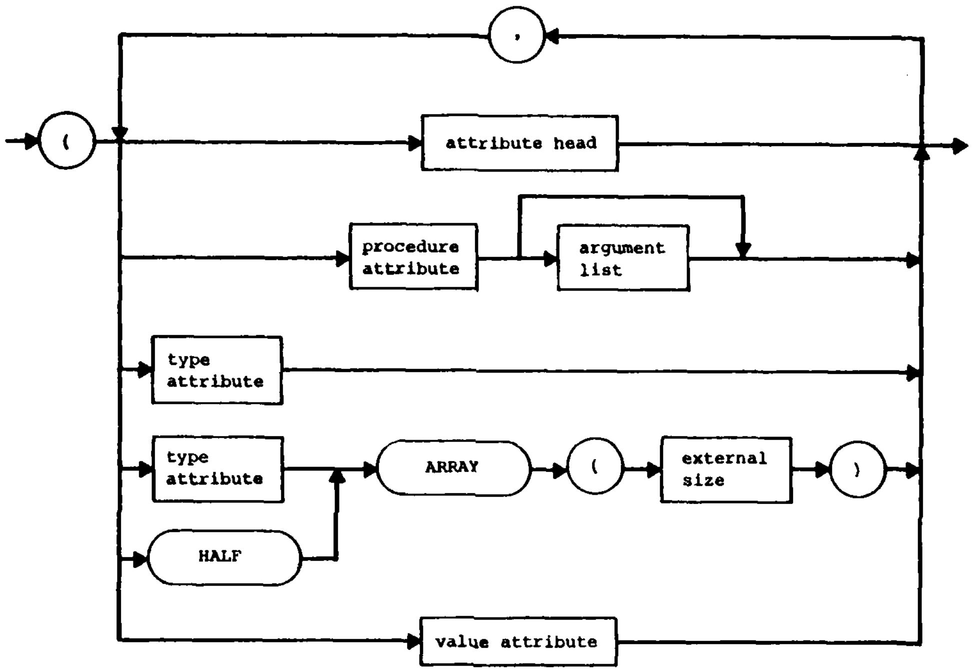


If the formal parameter procedure has arguments, they must be described in order of occurrence within parentheses. It is also necessary to indicate the transmission mode of these arguments.

argument list

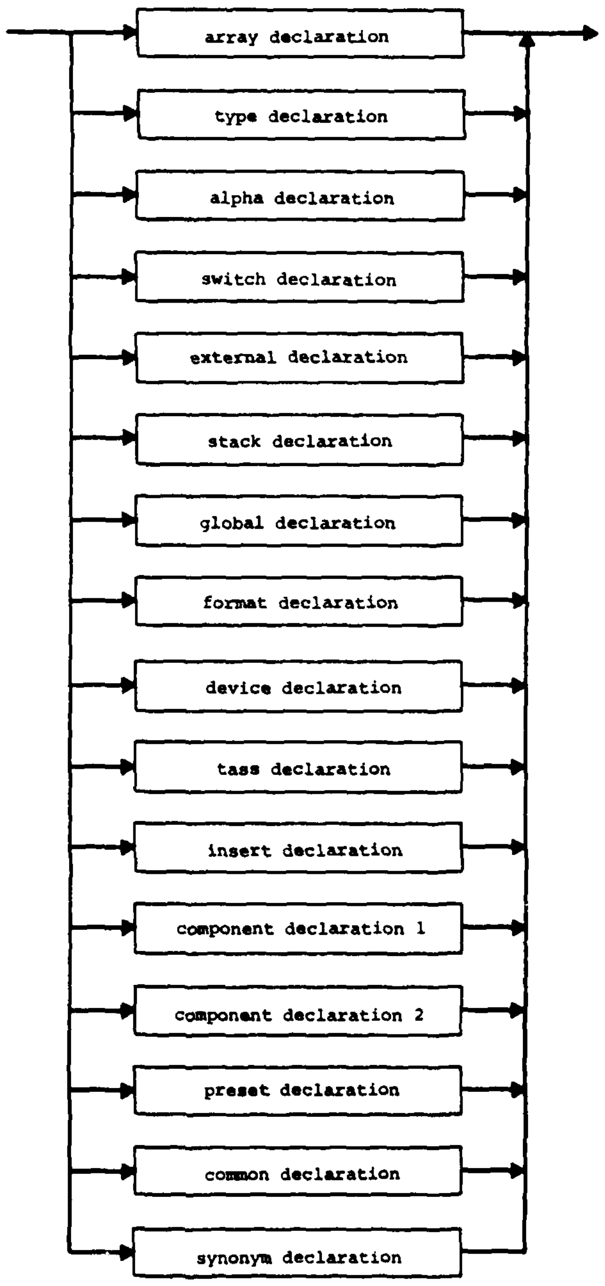


argument head



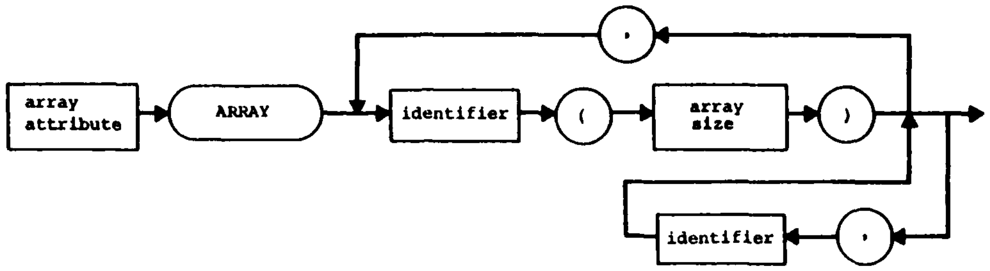
Data Declarations. All identifiers (symbols) used in a THLL program must be declared.

data declaration

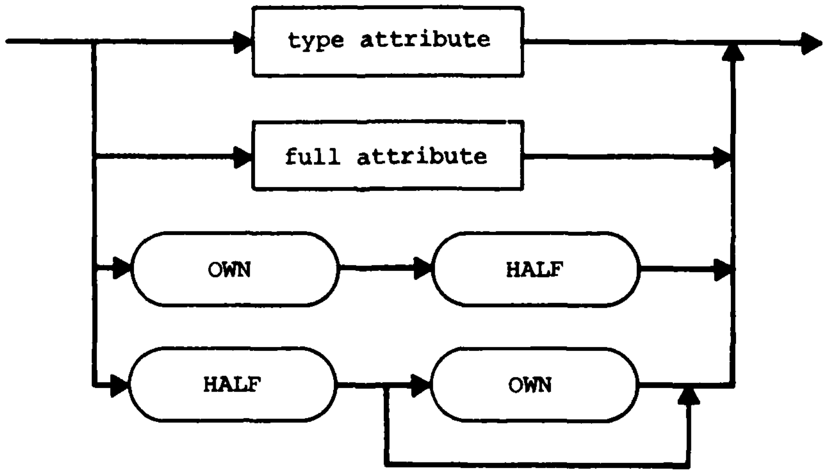


Array Declarations. Array declarations provide information concerning the type, name, and size of an array, as well as indicating its allocation mode. Array subscripting starts with zero. For example, an array named TAB which is to be one-dimensional and contain 10 elements would be indicated by TAB(9). Arrays may be one-, two-, or three-dimensional with the maximum size of each dimension being specified by an integer constant in the declaration statement.

array declaration

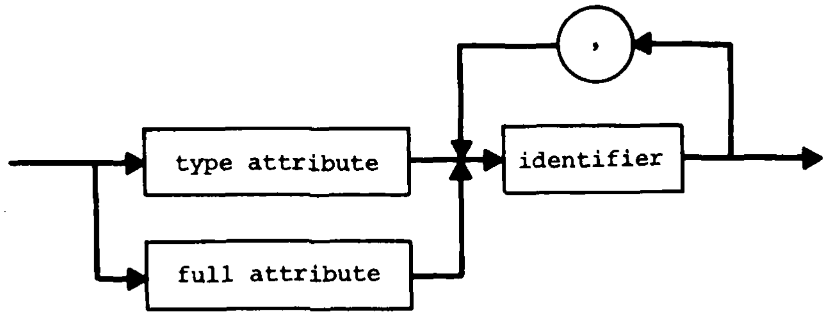


array attribute



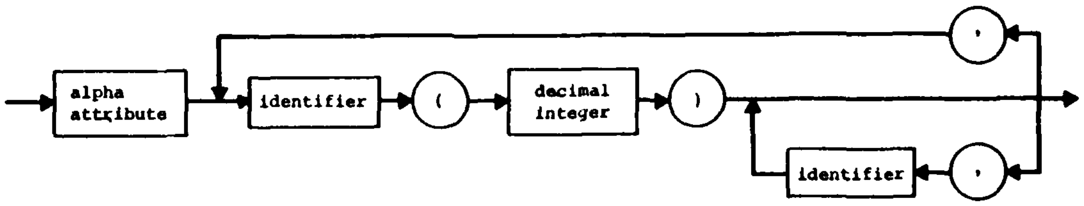
Type Declarations. Type declarations specify variables to be INTEGER, DOUBLE, REAL, or POINTER. The variable specified by an identifier can only assume values of the declared type by assignment or preset declaration. If the variable receives an assigned value, automatic type conversion will have occurred.

type declaration

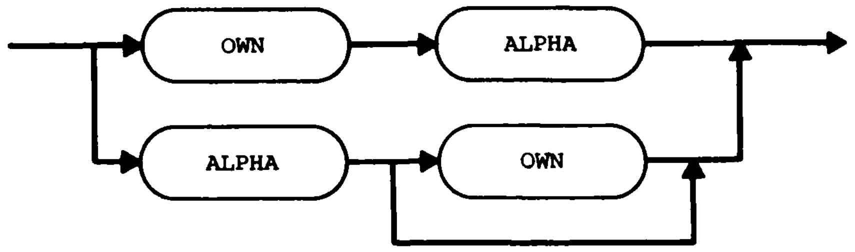


Alpha Declarations. An alpha declaration defines an identifier to be of type ALPHA. The maximum length of any string that can be stored is specified in the declaration. If more than one string is declared in the same declaration, all will have the same length unless indicated otherwise. The first character of a string is designated as the zero character.

alpha declaration



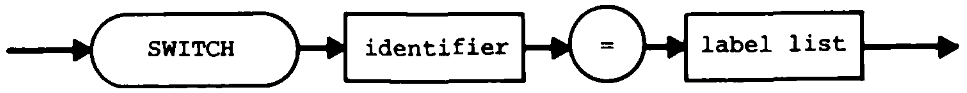
alpha attribute



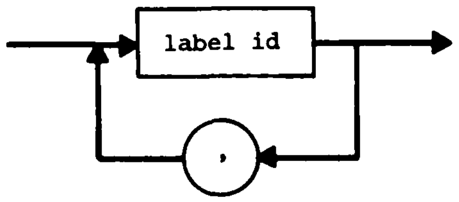
Switch Declarations. The identifier following the reserved word, SWITCH, is associated with a sequence of labels to the right of the assignment operator. A positive integer, indicating relative position within the list from left to right, is associated with each label identifier.

If "L", an identifier, is bound to a switch list by a switch declaration then the value of the address expression L(e) is the nth label in the switch list for "L", where n is the integer value of e. If n is less than 1 or n is greater than the count for the last label, then n is set to the value of the last label.

switch declaration

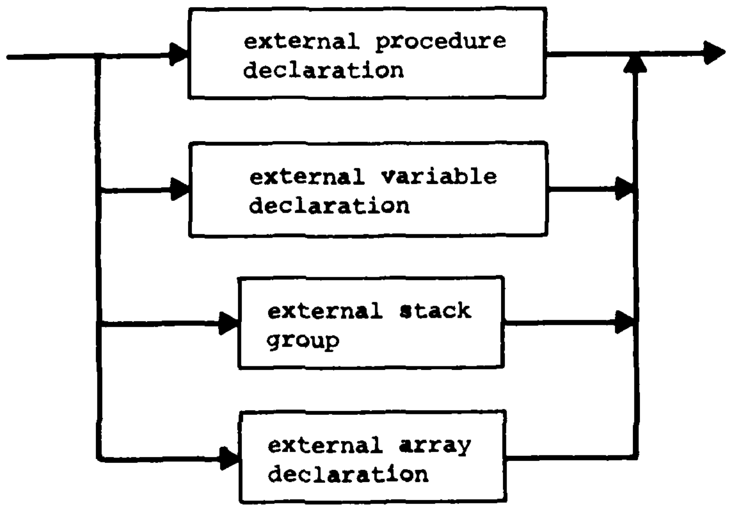


label list

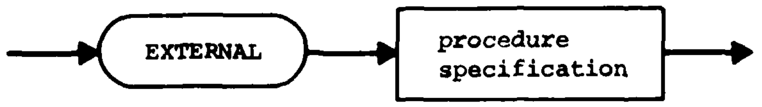


<u>External Declarations</u>. An external declaration does not define an identifier, it equivalences it to an identifier defined outside of the program. Sufficient information about the identifier, its type, its dimensions if it is an array, its arguments if it is a procedure, must be given to allow the identifier declared EXTERNAL to be treated properly by the compiler. Component, label, and switch identifiers cannot be declared EXTERNAL.

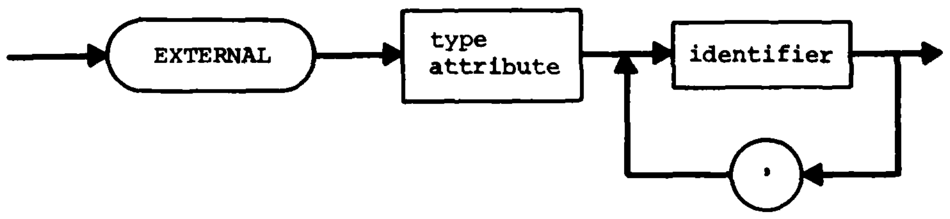
<u>external declaration</u>


external procedure declaration
external variable declaration
external stack group
external array declaration

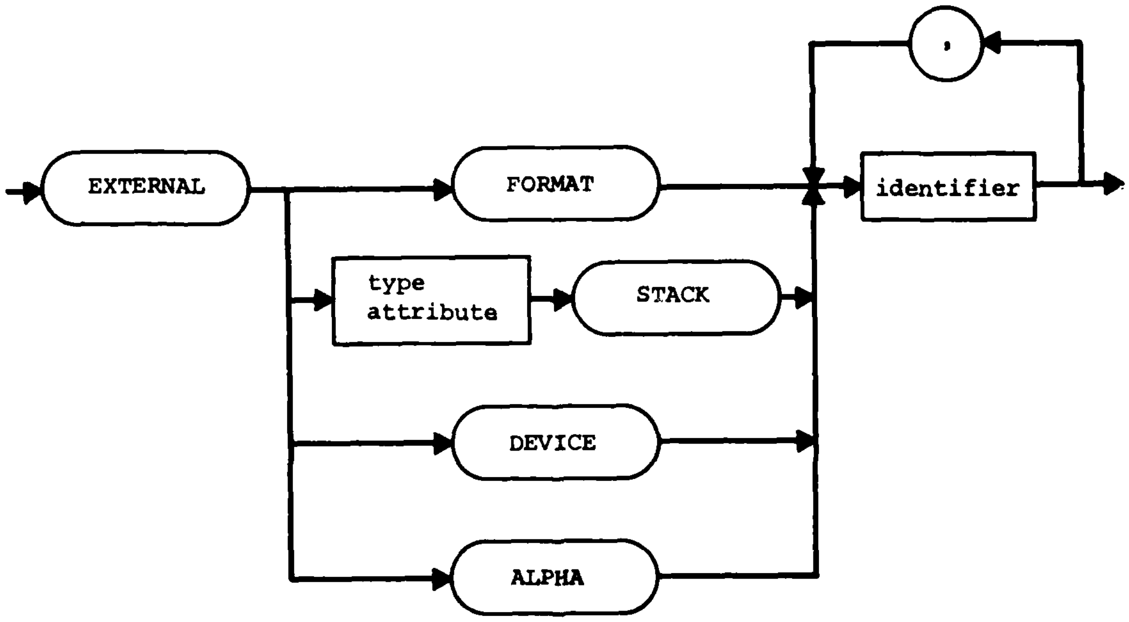
<u>external procedure declaration</u>



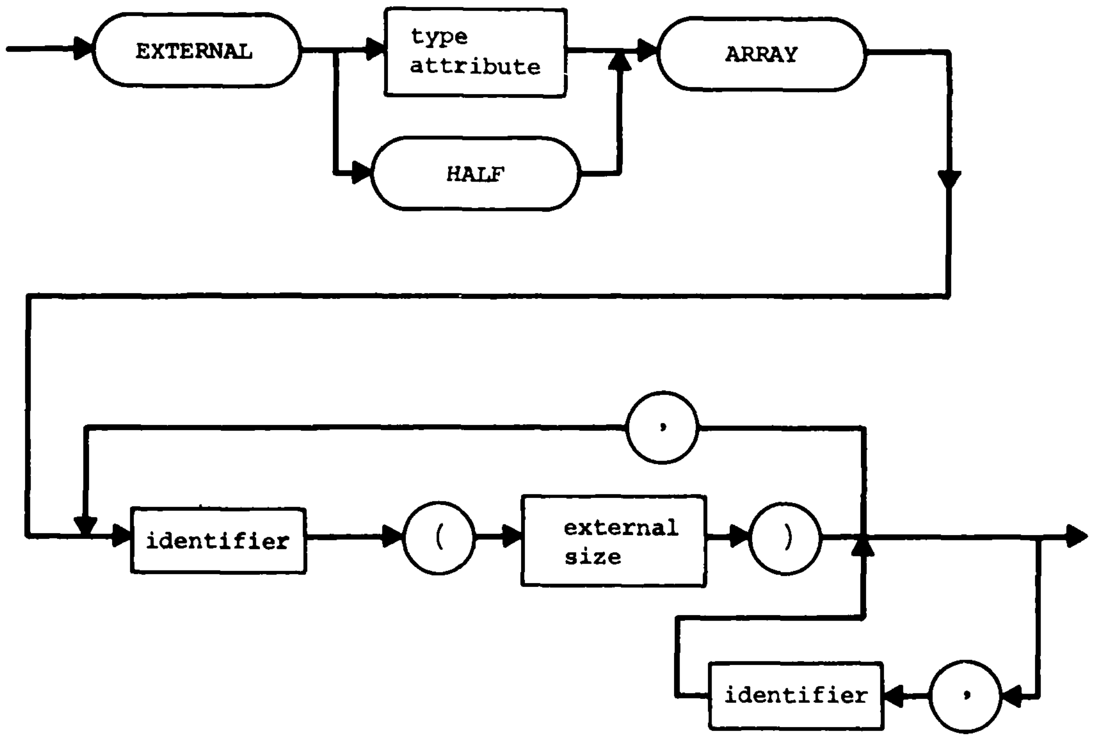
external variable declaration



external stack group

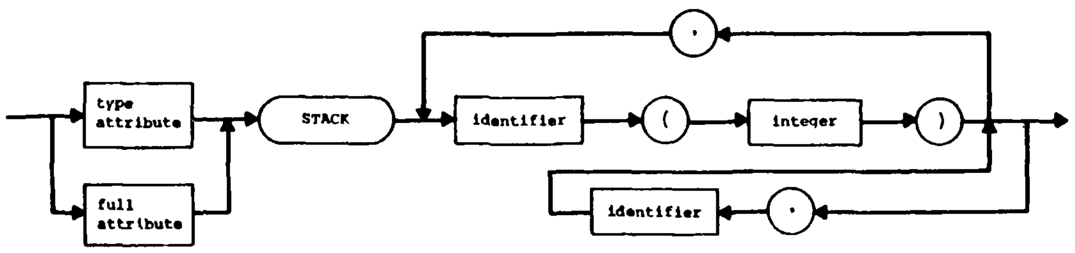


external array declaration



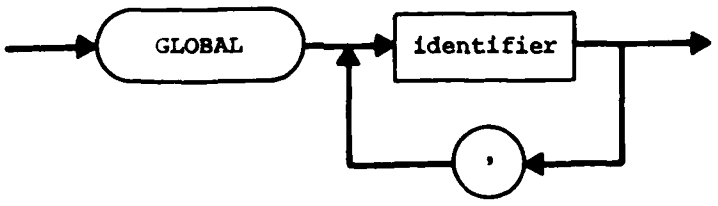
Stack Declarations. Stack declarations provide information concerning the type, name, and size of a stack, as well as indicating its allocation mode.

stack declaration



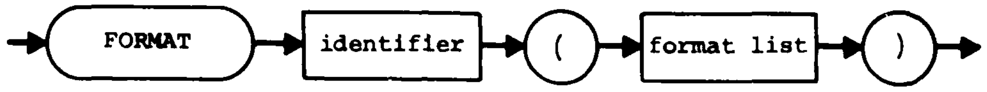
Global Declaration. A global declaration makes an identifier, defined in one program, known to a separately compiled program.

global declaration

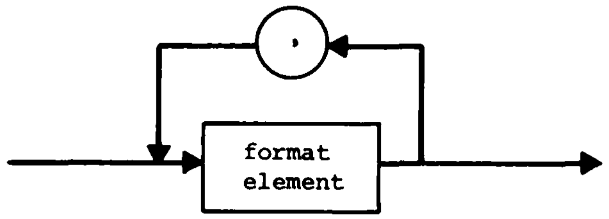


Format Declaration. A format declaration binds an identifier to a format list which is used to indicate the manner of converting and editing information between the internal representation and the external character string.

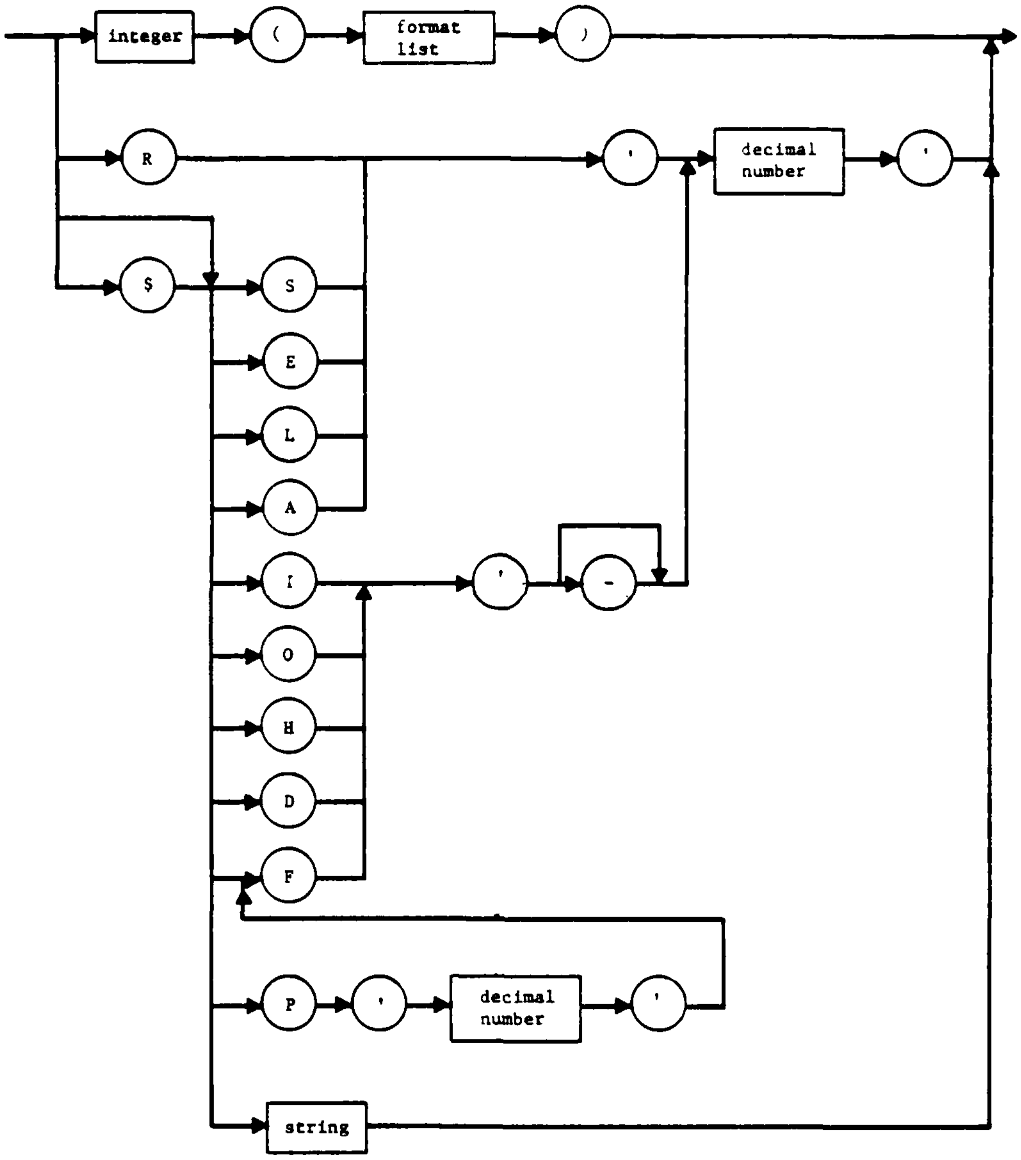
format declaration



format list

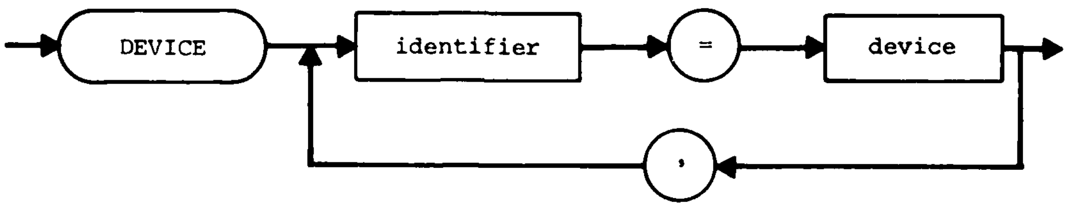


format element

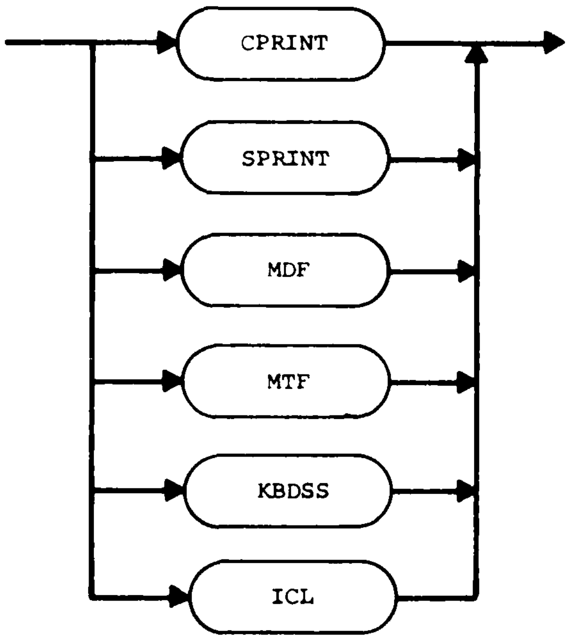


Device Declarations. In a device declaration, the identifier is bound to the TDCC hardware device specified.

device declaration

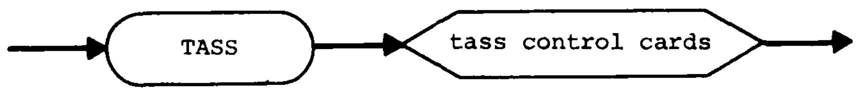


device



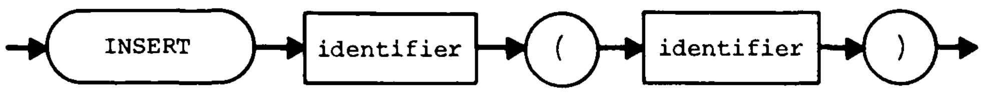
TASS Declaration. The TASS declaration permits the programmer to include TASS control cards as a part of the THLL source text. The TASS declaration can appear anywhere in the source text where a declaration can appear. Ideally, it should appear immediately after the first BEGIN. The effect of the TASS declaration is to treat all card images, beginning with the next card until a card image with a double slash (i.e., //) in card columns 1 and 2 is encountered, as TASS control.

TASS declaration



INSERT Declarations. The INSERT declaration permits the programmer to include source text from other than the current compiler source. The source text to be included is selected from a data set referenced by file name (first identifier) and member name (second identifier).

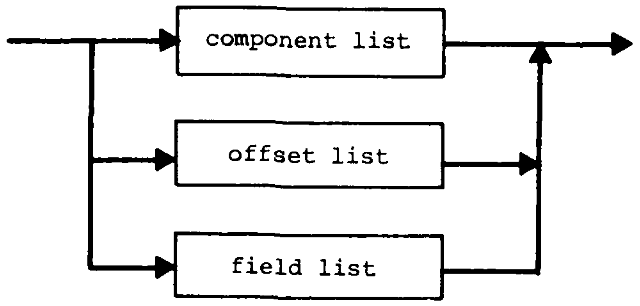
INSERT declaration



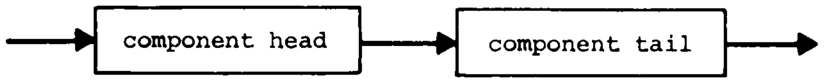
Component Declarations. By using components, a programmer is able to access parts of a word. Also, a specific field of a word may be written into by using a component as the left-side variable in an assignment expression.

Component declarations come in two forms. One form allows all component characteristics to be specified for one symbol in a single declaration, while the other form allows the type information, the field information, and the offset information to be specified for a list of component identifiers by separate declarations.

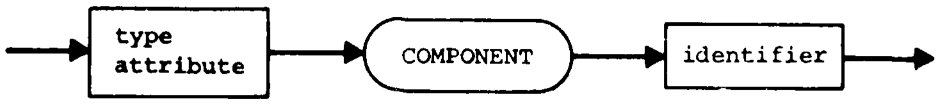
component declaration 1



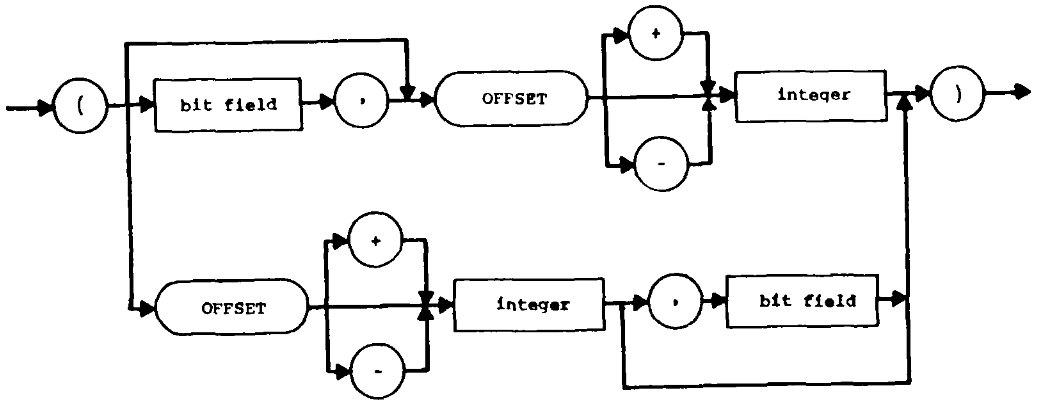
component declaration 2



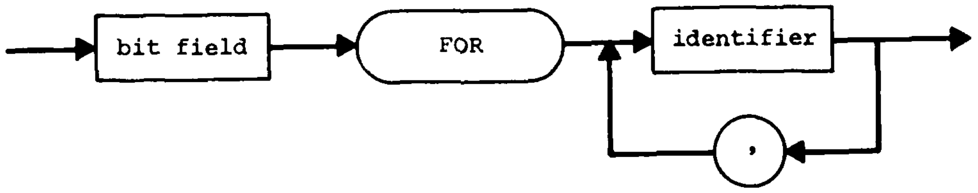
component head



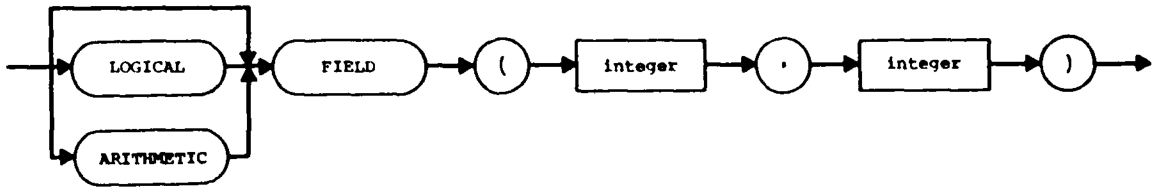
component tail



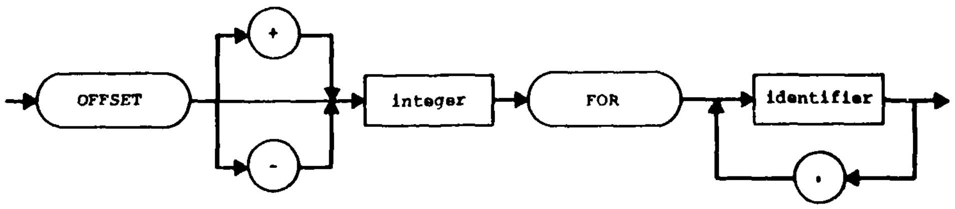
field list



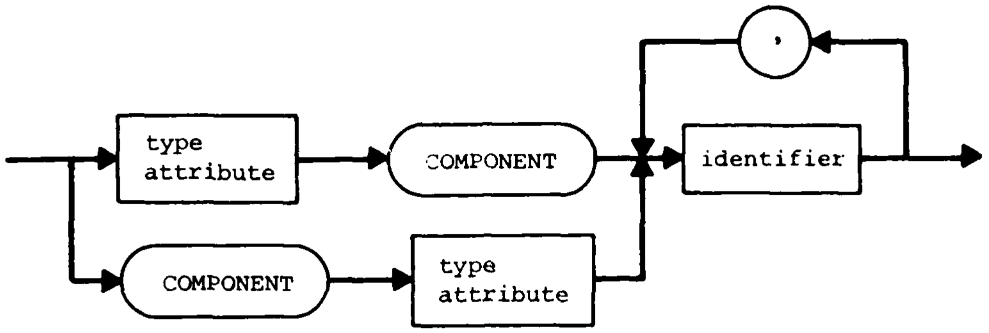
bit field



offset list



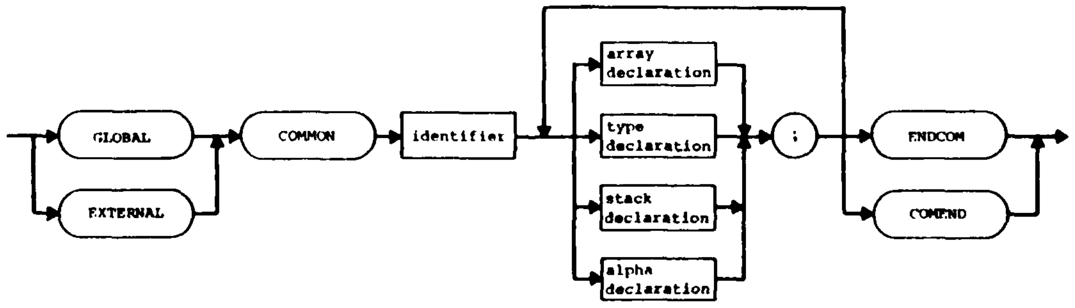
component list



If an identifier is declared to be a component according to component declaration 1, and the component field is a proper part of a word, then this identifier must appear in all three lists; the component list, the offset list, and the field list. For whole word and double word components the field definition may be omitted.

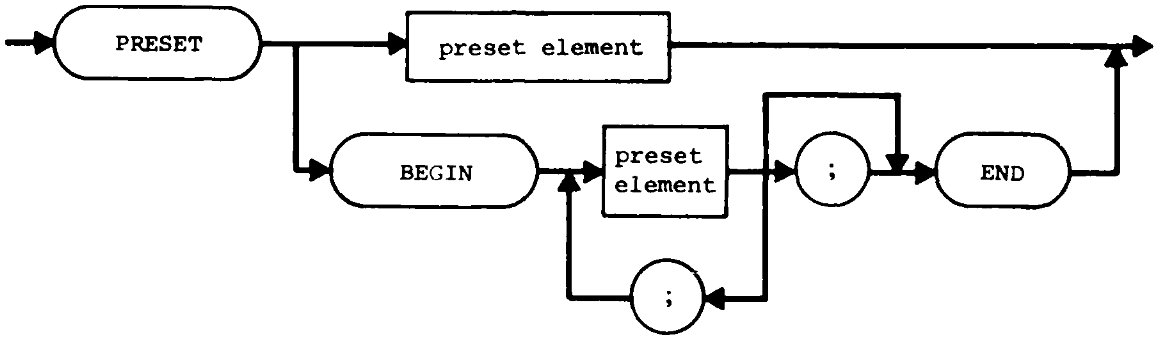
Common Declarations. The common declaration provides a mechanism for defining a block of memory for OWN data such that only the origin of this block is a global symbol and that various parts of this block can be referenced symbolically. The common body, that is, the portion of the common declaration following the keyword COMMON, should not contain the keyword OWN. All common data are implied to be OWN. With this agreement, the common body is identical in the GLOBAL COMMON definition and in the EXTERNAL COMMON use.

common declaration

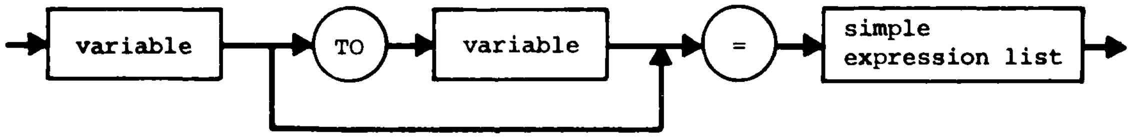


Preset Declaration. The preset declaration is used to initialize OWN variables at compile time. Simple and subscripted variables may be preset. Subscripted variables using a stack identifier cannot be preset.

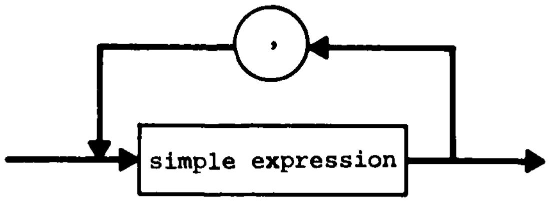
preset declaration



preset element



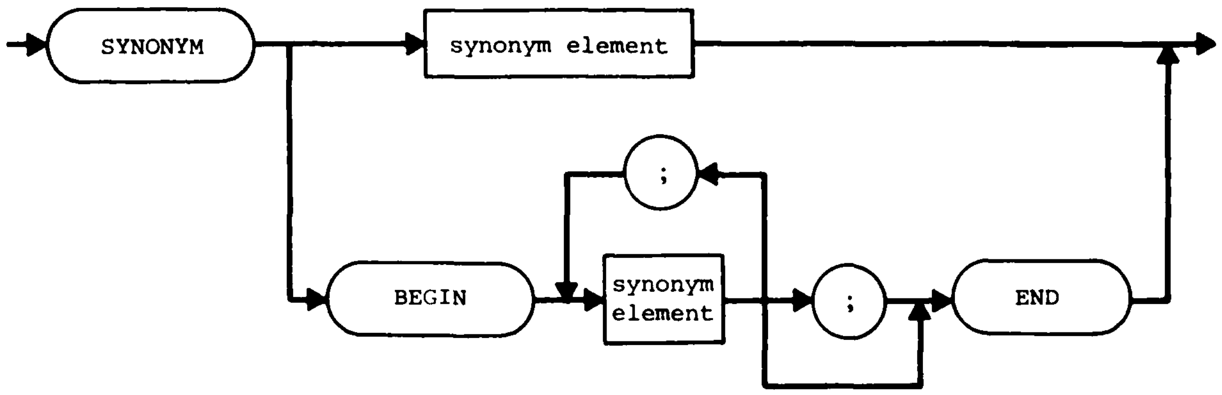
simple expression list



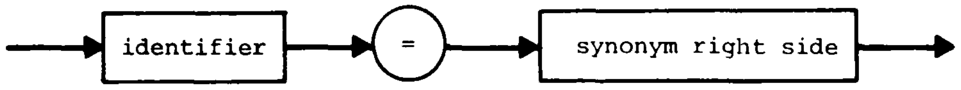
Only expressions that can be evaluated at compile time must appear in the simple expression list.

Synonym Declaration. A synonym declaration allows a THLL programmer to associate a segment of source text with an identifier. When dollar signs ($) are used to delimit a synonym definition, the synonym definition may not contain any imbedded dollar signs.

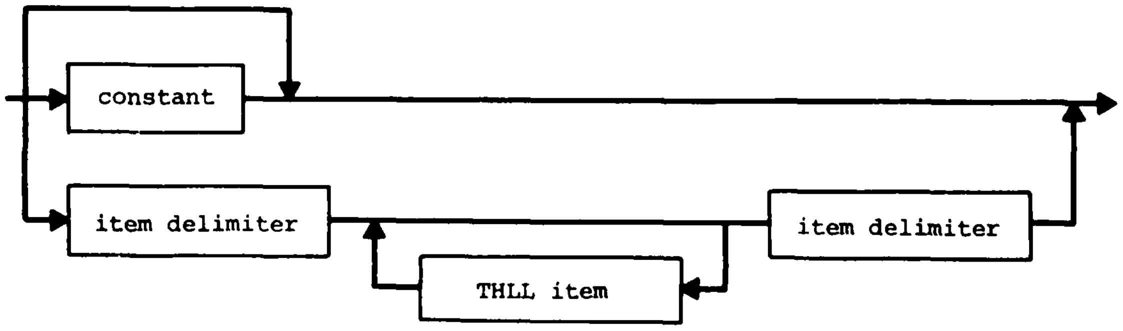
synonym declaration



synonym element



synonym right side

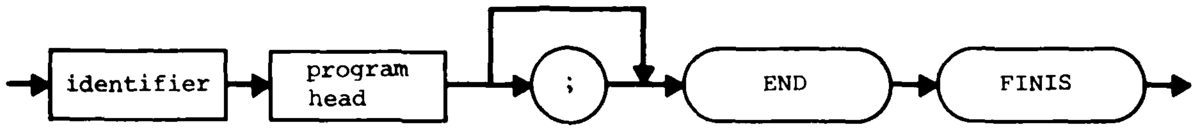


A THLL item is a constant, an identifier, an operator or a delimiter. An item delimiter is a THLL item that is not a constant and not the semicolon. The right-end item delimiter is the first occurrence of the source THLL item that functions as the left-end item delimiter.

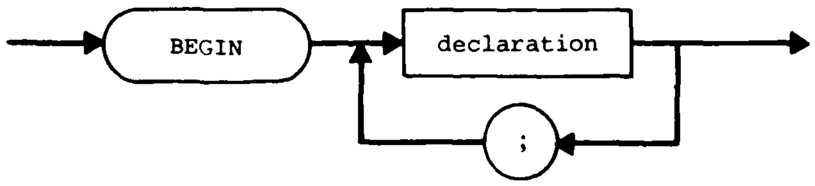
PROGRAMS

A THLL program is simply a BEGIN-END FINIS block containing one or more declarations.

program



program head



APPENDIX A

BNF DEFINITION OF THLL

```
                         BNF GRAMMAR FOR THLL
                         --- ------- --- ----

    1.  BASIC SYMBCLS, CONSTANTS, AND IOFNTIFIERS
        -----------------------------------------

        1.1  CHARACTFRS
             ----------

          1. <LETTFR>!!= A\B\C\D\E\F\G\H\I\J\K\L\M\N\O\P\Q\R\S\T\U\V\W\X\Y\Z

          2. <DIGIT>!!=  0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9

          3. <SPFCIAL CHARACTER>!!= -\+\*\/\(\)\=\,\.\:\;\#\"\$\!\

                                    '\<\>\%\_\ \&\?\@\[\]\\\^

          4. <CHARACTER>!!= <LETTER> \ <DIGIT> \ <SPECIAL CHARACTER>

        1.2  CONSTANTS
             ---------

          5. <CONSTANT>!!=  <NUMBER> \ <BOOLEAN CONSTANT> \ <STRING>

        1.2.1  NUMBERS
               -------

          6. <NUMBER>!!=  <INTEGER> \ <REAL NUMBER> \ <SCALED REAL NUMBER>

        1.2.1.1  INTEGERS
                 --------

          7. <BINARY DIGIT>!!=  0\1

          8. <OCTAL DIGIT>!!=  <BINARY DIGIT>\2\3\4\5\6\7

          9. <DECIMAL DIGIT>!!=  <OCTAL DIGIT>\8\9

         10. <HEX DIGIT>!!=  <DECIMAL DIGIT>\A\B\C\D\E\F

         11. <DECIMAL INTEGER>!!=  <DECIMAL DIGIT> \

                                   <DECIMAL INTEGER> <DECIMAL DIGIT>

         12. <SCALE PART>!!=  K <DECIMAL INTEGER> \ K+ <DECIMAL INTEGER> \

                              K- <DECIMAL INTEGER>

         13. <BINARY INTEGER>!!=  <BINARY DIGIT> \ <BINARY INTEGER> <BINARY DIGIT>

         14. <BINARY NUMBER>!!=  B'<OPTIONAL -> <BINARY INTEGER>' \

                                 B'<OPTIONAL -> <BINARY INTEGER> <SCALE PART>'
```

15. <OCTAL INTEGER>!!= <OCTAL DIGIT> \ <OCTAL INTEGER> <OCTAL DIGIT>

16. <OCTAL NUMBER>!!= C'<OPTIONAL -> <OCTAL INTEGER>' \
C'<OPTIONAL -> <OCTAL INTEGER> <SCALE PART>'

17. <DECIMAL NUMBER>!!= <DECIMAL INTEGER> \
<DECIMAL INTEGER> <SCALE PART>

18. <HEX INTEGER>!!= <HEX DIGIT> \ <HEX INTEGER> <HEX DIGIT>

19. <HEX NUMBER>!!= X'<OPTIONAL -> <HEX INTEGER>' \
X'<OPTIONAL -> <HEX INTEGER> <SCALE PART>'

20. <INTEGER>!!= <BINARY NUMBER> \ <OCTAL NUMBER> \ <DECIMAL NUMBER> \
<HEX NUMBER>

1.2.1.2 REAL NUMBERS

21. <EXPONENT>!!= E <DECIMAL INTEGER> \ E + <DECIMAL INTEGER> \
E - <DECIMAL INTEGER>

22. <FLOATING POINT NUMBER>!!= <DECIMAL INTEGER> . <DECIMAL INTEGER> \
<DECIMAL INTEGER> . \
<DECIMAL INTEGER>

23. <REAL NUMBER>!!= <FLOATING POINT NUMBER> \
<FLOATING POINT NUMBER> <EXPONENT>

1.2.1.3 SCALED REAL NUMBER

24. <SCALED REAL NUMBER>!!= <REAL NUMBER> <SCALE PART>

1.2.2 BOOLEAN CONSTANTS

25. <BOOLEAN CONSTANT>!!= TRUE \ FALSE

1.2.3 STRING

26. <CHARACTER SEQUENCE>!!= <EMPTY> \
<CHARACTER SEQUENCE> <CHARACTER>

27. <DELIMITING CHARACTER>!!= ANY CHARACTER NOT IN THE CHARACTER
SEQUENCE

```
28. <STRING>!!=  #<DELIMITING CHARACTER> <CHARACTER SEQUENCE>
                                                  <DELIMITING CHARACTER>
```

1.3 IDENTIFIERS

```
29. <PERIODS>!!=  . \ <PERIODS> .

30. <IDENTIFIER>!!=  <LETTER> \ <IDENTIFIER> <LETTER> \
                     <IDENTIFIER> <DECIMAL DIGIT> \
                     <IDENTIFIER> <PERIODS> <LETTER> \
                     <IDENTIFIER> <PERIODS> <DECIMAL DIGIT>

31. <COMPONENT ID>!!=  <IDENTIFIER>

32. <ARRAY ID>!!=  <IDENTIFIER>

33. <STACK ID>!!=  <IDENTIFIER>

34. <DEVICE ID>!!=  <IDENTIFIER>

35. <PROCEDURE ID>!!=  <IDENOIFIER>

36. <FORMAT ID>!!=  <IDENTIFIER>

37. <VARIABLE ID>!!=  <IDENTIFIER>

38. <LABEL ID>!!=  <IDENTIFIER>

39. <SWITCH ID>!!=  <IDENTIFIER>
```

2. EXPRESSIONS

```
40. <EXPRESSION>!!=  <SIMPLE EXPRESSION> \ <ASSIGNMENT EXPRESSION>
```

2.1 VARIABLES

```
41. <COMPONENT HEAD>!!=  <COMPONENT ID> ( <EXPRESSION> \
                         <COMPONENT HEAD> , <EXPRESSION>

42. <COMPONENT VARIABLE>!!=  <COMPONENT HEAD> )

43. <ARRAY HEAD>!!=  <ARRAY ID> ( <EXPRESSION> \
                     <ARRAY HEAD> , <EXPRESSION>

44. <SUBSCRIPTED VARIABLE>!!=  <ARRAY HEAD> ) \
                               <STACK ID> ( <EXPRESSION> )

45. <VARIABLE>!!=  <VARIABLE ID> \ <SUBSCRIPTED VARIABLE> \
                   <COMPONENT VARIABLE>
```

2.2 FUNCTION DESIGNATORS

```
46. <ACTUAL PARAMETER>!!=  <EXPRESSION> \ <ARRAY ID> \ <STACK ID> \
                           <DEVICE ID> \ ENTRYP <PROCEDURE ID> \
                           <FORMAT ID> \ <LOOP ARGUMENT>

47. <PARAMETER LIST>!!=  <ACTUAL PARAMETER> \
                         <PARAMETER LIST> , <ACTUAL PARAMETER>

48. <LOOP STEP HEAD>!!=  FOR <VARIABLE ID> = <EXPRESSION> STEP
                                                          <EXPRESSION>

49. <LOOP REPEAT HEAD>!!=  FOR <VARIABLE ID> = <EXPRESSION> REPEAT
                                                          <EXPRESSION>

50. <LOOP ARGUMENT TAIL>!!=  <EXPRESSION> ( <PARAMETER LIST> )

51. <LOOP ARGUMENT>!!=  WHILE <LOOP ARGUMENT TAIL> \
                        <LOOP STEP HEAD> WHILE <LOOP ARGUMENT TAIL> \
                        <LOOP STEP HEAD> UNTIL <LOOP ARGUMENT TAIL> \
                        <LOOP REPEAT HEAD> WHILE <LOOP ARGUMENT TAIL>
```

```
52. <FUNCTION DESIGNATOR>!!=  <PROCEDURE ID> \
                              <PROCEDURE ID> ( <PARAMETER LIST> )
```

2.3 SIMPLE EXPRESSIONS

```
53. <RELATIONAL OPERATOR>!!=  LES \ LEQ \ EQL \ GRT \ GEQ \ NEQ

54. <RELATION>!!=  <ARITHMETIC EXPRESSION> <RELATIONAL OPERATOR>
                                              <ARITHMETIC EXPRESSION>

55. <BOOLEAN PRIMARY>!!=  <RELATION> \ <ARITHMETIC EXPRESSION>

56. <BOOLEAN SECONDARY>!!=  <BOOLEAN PRIMARY> \ NOT <BOOLEAN PRIMARY>

57. <BOOLEAN FACTOR>!!=  <BOOLEAN SECONDARY> \
                         <BOOLEAN FACTOR> AND <BOOLEAN SECONDARY>

58. <BOOLEAN TERM>!!=  <BOOLEAN FACTOR> \
                       <BOOLEAN TERM> OR <BOOLEAN FACTOR> \
                       <BOOLEAN TERM> XOR <BOOLEAN FACTOR>

59. <BOOLEAN EXPRESSION>!!=  <BOOLEAN TERM>

60. <SIMPLE EXPRESSION>!!=  <BOOLEAN EXPRESSION>
```

2.3.1 ARITMETIC EXPRESSIONS

```
61. <NOT>!!=  BITNOT \ NOTB

62. <AND>!!=  BITAND \ ANDB

63. <OR>!!=  BITOR \ ORB

64. <XOR>!!=  BITXOR \ XORB

65. <MULT OP>!!=  * \ / \ MOD

66. <ADD OP>!!=  + \ -

67. <PRIMARY>!!=  <CONSTANT> \ <VARIABLE> \ LOC <VARIABLE> \
                  LOC <PROCEDURE ID> \ LOC <FORMAT ID> \
                  <FUNCTION DESIGNATOR> \ <CONDITIONAL EXPRESSION> \
                  <CASE EXPRESSION> \ ( <EXPRESSION> )

68. <PRIMARY 2>!!=  <PRIMARY> \ <NOT> PRIMARY

69. <PRIMARY 3>!!=  <PRIMARY 2> \ <PRIMARY 3> <AND> <PRIMARY 2>
```

```
70. <PRIMARY 4>!!=  <PRIMARY 3> \
                    <PRIMARY 4> <OR> <PRIMARY 3> \
                    <PRIMARY 4> <XOR> <PRIMARY 3>
71. <FACTOR>!!=  <PRIMARY 4> \ <FACTOR> ** <PRIMARY 4>
72. <TERM>!!=  <FACTOR> \ <TERM> <MULT OP> <FACTOR>
73. <ARITHMETIC EXPRESSION>!!=  <TERM> \ <ADD OP> <TERM> \
                                <ARITHMETIC EXPRESSION> <ADD OP> <TERM>
```

2.3.2 CONDITIONAL EXPRESSION

```
74. <IF HEAD>!!=  IF <EXPRESSION> THEN <EXPRESSION> \
                  IF <EXPRESSION> THEN <STATEMENT> \
                  <IF HEAD> , <EXPRESSION> THEN <EXPRESSION> \
                  <IF HEAD> , <EXPRESSION> THEN <STATEMENT>
75. <IF TAIL>!!=  IFEND \ ENDIF
76. <CONDITIONAL EXPRESSION>!!=  <IF HEAD> <IF TAIL> \
                                 <IF HEAD> ELSE <EXPRESSION> <IF TAIL> \
                                 <IF HEAD> ELSE <STATEMENT> <IF TAIL>
```

2.3.3 CASE EXPRESSION

```
77. <CASE HEAD>!!=  CASE <EXPRESSION> DO <EXPRESSION> \
                    CASE <EXPRESSION> DO <STATEMENT> \
                    <CASE HEAD> , <EXPRESSION> \
                    <CASE HEAD> , <STATEMENT>
78. <CASE TAIL>!!=  CASEEND \ ENDCASE
79. <CASE EXPRESSION>!!=  <CASE HEAD> <CASE TAIL>
```

2.4 ASSIGNMENT EXPRESSIONS

```
80. <ASSIGNMENT EXPRESSION>!!=  <VARIABLE> = <EXPRESSION>
```

3. STATEMENTS

```
81. <STATEMENT>!!=  <CHANGE CONTROL STATEMENT> \ <PROPER STATEMENT>
```

3.1 CHANGE OF CONTROL STATEMENTS

```
82. <CHANGE CONTROL STATEMENT>!!=  <GOTO STATEMENT> \ <EXIT STATEMENT> \
                                   <RETURN STATEMENT>
```

3.1.1 GOTO STATEMENTS

```
83. <GOTO STATEMENT>!!=  GOTO <LABEL ID> \
                         GOTO <SWITCH ID> ( <EXPRESSION> )
```

3.1.2 EXIT STATEMENTS

```
84. <EXIT STATEMENT>!!=  EXIT \ EXIT <LABEL ID> \
                         LOOPEXIT \ LOOPEXIT <LABEL ID>
```

3.1.3 RETURN STATEMENTS

```
85. <RETURN STATEMENT>!!=  RETURN \ RETURN <EXPRESSION>
```

3.2 PROPER STATEMENTS

```
86. <PROPER STATEMENT>!!=  <BLOCK STATEMENT> \ <LOOP STATEMENT> \
                           <NULL STATEMENT>
```

3.2.1 BLOCK STATEMENTS

```
87. <LABEL END>!!=  <LABEL ID> : END \ <LABEL ID> : <LABEL END>
88. <LABEL STATEMENT>!!=  <EXPRESSION> \ <STATEMENT> \
                          <LABEL ID> : <LABEL STATEMENT>
89. <BLOCK HEAD>!!=  <PROGRAM HEAD> ; <LABEL STATEMENT> \
                     BEGIN <LABEL STATEMENT> \
                     <BLOCK HEAD> ; <LABEL STATEMENT>
```

```
90. <BLOCK STATEMENT>!!=  <BLOCK HEAD> END \ <BLOCK HEAD> ; END \
                          <BLOCK HEAD> ; <LABEL END>
```

3.2.2 LOOP STATEMENTS
---- ----------

```
91. <LOOP STATEMENT TAIL>!!=  <EXPRESSION> DO <EXPRESSION> \
                              <EXPRESSION> DO <PROPER STATEMENT>
92. <LOOP STATEMENT>!!=  WHILE <LOOP STATEMENT TAIL> \
                        <LOOP STEP HEAD> UNTIL <LOOP STATEMENT TAIL> \
                        <LOOP STEP HEAD> WHILE <LOOP STATEMENT TAIL> \
                        <LOOP REPEAT HEAD> WHILE <LOOP STATEMENT TAIL>
```

3.2.3 NULL STATEMENT

```
93. <NULL STATEMENT>!!=  NULL
```

4. DECLARATIONS

```
 94. <DECLARATION>!!=  <PROCEDURE DECLARATION> \ <DATA DECLARATION>
```

4.1 ATTRIBUTES

```
 95. <TYPE ATTRIBUTE>!!=  REAL \ INTEGER \ DOUBLE \ POINTER

 96. <FULL ATTRIBUTE>!!=  OWN <TYPE ATTRIBUTE> \ <TYPE ATTRIBUTE> OWN

 97. <ATTRIBUTE HEAD>!!=  <TYPE ATTRIBUTE> STACK \
                          FORMAT \ ALPHA \ DEVICE

 98. <VALUE ATTRIBUTE>!!=  VALUE <TYPE ATTRIBUTE> \
                           <TYPE ATTRIBUTE> VALUE

 99. <PROCEDURE ATTRIBUTE>!!=  PROCEDURE \ <TYPE ATTRIBUTE> PROCEDURE \
                               LINK PROCEDURE \
                               <TYPE ATTRIBUTE> LINK PROCEDURE

100. <ARRAY SIZE>!!=  <INTEGER> \ <INTEGER> , <INTEGER> \
                      <INTEGER> , <INTEGER> , <INTEGER>

101. <EXTERNAL SIZE>!!=  <ARRAY SIZE> \ * \ * , * \ * , * , *
```

4.2 PROCEDURE DECLARATIONS

```
102. <ARGUMENT HEAD 2>!!=  <ATTRIBUTE HEAD> \ <TYPE ATTRIBUTE> \
                           <PROCEDURE ATTRIBUTE> \
                           <PROCEDURE ATTRIBUTE> <ARGUMENT LIST> \
                           <TYPE ATTRIBUTE> ARRAY ( <EXTERNAL SIZE> ) \
                           HALF ARRAY ( <EXTERNAL SIZE> ) \
                           <VALUE ATTRIBUTE>

103. <ARGUMENT HEAD>!!=  ( <ARGUMENT HEAD 2> \
                         <ARGUMENT HEAD> , <ARGUMENT HEAD 2>

104. <ARGUMENT LIST>!!=  <ARGUMENT HEAD> ) \ ( OPTARG ) \
                         <ARGUMENT HEAD> , OPTARG )
```

```
105. <ARRAY TYPE>!!= <TYPE ATTRIBUTE> ARRAY \ HALF ARRAY
106. <ARRAY SPECIFICATION>!!=  <ARRAY TYPE> <IDENTIFIER>
                                                    ( <EXTERNAL SIZE> ) \
                               <ARRAY SPECIFICATION> , <IDENTIFIER> \
                               <ARRAY SPECIFICATION> , <IDENTIFIER>
                                                    ( <EXTERNAL SIZE> )
107. <PROCEDURE SPECIFICATION>!!=  <PROCEDURE ATTRIBUTE> <IDENTIFIER> \
                                   <PROCEDURE ATTRIBUTE> <IDENTIFIER>
                                                   <ARGUMENT LIST>
108. <SPECIFICATION ELEMENT>!!=  <ATTRIBUTE HEAD> <IDENTIFIER> \
                                 <TYPE ATTRIBUTE> <IDENTIFIER> \
                                 <SPECIFICATION ELEMENT> , <IDENTIFIER>
109. <SPECIFICATION PART>!!=  <SPECIFICATION ELEMENT> \
                              <PROCEDURE SPECIFICATION> \
                              <ARRAY SPECIFICATION>
110. <EXECUTIVE HEAD>!!=  EXEC \ EXEC INTERRUPT <INTEGER>
111. <DESCRIPTOR HEAD>!!=  <EXECUTIVE HEAD> PROCEDURE <IDENTIFIER> \
                           <PROCEDURE ATTRIBUTE> <IDENTIFIER> \
                           <PROCEDURE ATTRIBUTE> <IDENTIFIER> ( OPTARG )
112. <PROCEDURE HEAD 1>!!=  DEFINE <PROCEDURE ATTRIBUTE> <IDENTIFIER>
                                                      ( <IDENTIFIER> \
                            <PROCEDURE HEAD 1> , <IDENTIFIER>
113. <PROCEDURE HEAD 2>!!=  <PROCEDURE HEAD 1> ) \
                            <PROCEDURE HEAD 1> , OPTARG )
114. <PROCEDURE HEAD 3>!!=  <PROCEDURE HEAD 2> ; VALUE <IDENTIFIER> \
                            <PROCEDURE HEAD 3> , <IDENTIFIER>
115. <PROCEDURE HEAD>!!=  DEFINE <DESCRIPTOR HEAD> \
                          <PROCEDURE HEAD 2> ; <SPECIFICATION PART> \
                          <PROCEDURE HEAD 3> ; <SPECIFICATION PART> \
                          <PROCEDURE HEAD> ; <SPECIFICATION PART>
```

```
116. <PROCEDURE BODY>!!=  <BLOCK STATEMENT>

117. <PROCEDURE DECLARATION>!!=  <PROCEDURE HEAD> ; <PROCEDURE BODY>
```

4.3 DATA DECLARATIONS

```
118. <DATA DECLARATION>!!=  <ARRAY DECLARATION> \ <TYPE DECLARATION> \
                            <ALPHA DECLARATION> \ <SWITCH DECLARATION> \
                            <EXTERNAL DECLARATION> \
                            <STACK DECLARATION> \ <GLOBAL DECLARATION> \
                            <FORMAT DECLARATION> \ <DEVICE DECLARATION> \
                            <TASS DECLARATION> \ <INSERT DECLARATION> \
                            <COMPONENT DECLARATION 1> \
                            <COMPONENT DECLARATION 2> \
                            <COMMON DECLARATION> \ <PRESET DECLARATION> \
                            <SYNONYM DECLARATION>
```

4.3.1 ARRAY DECLARATIONS

```
119. <ARRAY ATTRIBUTE>!!=  <TYPE ATTRIBUTE> \ <FULL ATTRIBUTE> \
                          OWN HALF \ HALF \ HALF OWN
120. <ARRAY DECLARATION>!!=  <ARRAY ATTRIBUTE> ARRAY <IDENTIFIER>
                                                      ( <ARRAY SIZE> ) \
                             <ARRAY DECLARATION> , <IDENTIFIER> \
                             <ARRAY DECLARATION> , <IDENTIFIER>
                                                      ( <ARRAY SIZE> )
```

4.3.2 TYPE DECLARATIONS

```
121. <TYPE DECLARATION>!!=  <TYPE ATTRIBUTE> <IDENTIFIER> \
                            <FULL ATTRIBUTE> <IDENTIFIER> \
                            <TYPE DECLARATION> , <IDENTIFIER>
```

4.3.3 ALPHA DECLARATIONS

```
122. <ALPHA ATTRIBUTE>!!=  ALPHA \ OWN ALPHA \ ALPHA OWN

123. <ALPHA DECLARATION>!!=  <ALPHA ATTRIBUTE> <IDENTIFIER>
                                                  ( <INTEGER> ) \
                             <ALPHA DECLARATION> , <IDENTIFIER> \
                             <ALPHA DECLARATION> , <IDENTIFIER>
                                                    ( <INTEGER> )
```

4.3.4 SWITCH DECLAPATIONS

```
124. <LABEL LIST>!!=  <LABEL ID> \ <LABEL LIST> , <LABEL ID>

125. <SWITCH DECLARATION>!!=  SWITCH <IDENTIFIER> = <LABEL LIST>
```

4.3.5 EXTERNAL DECLARATIONS

```
126. <EXTERNAL DECLARATION>!!=  <EXTERNAL PROCEDURE DECLARATION> \
                                <EXTERNAL VARIABLE DECLARATION> \
                                <EXTERNAL STACK GROUP> \
                                <EXTERNAL ARRAY DECLARATION>
```

4.3.5.1 EXTERNAL PROCEDURE DECLARATIONS

```
127. <EXTERNAL PROCEDURE DECLARATION>!!=  EXTERNAL
                                          <PROCEDURE SPECIFICATION>
```

4.3.5.2 EXTERNAL VARIABLE DECLARATIONS

```
128. <EXTERNAL VARIABLE DECLARATION>!!=  EXTERNAL <TYPE ATTRIBUTE>
                                                        <IDENTIFIER> \
                                         <EXTERNAL VARIABLE DECLARATION>
                                                      <IDENTIFIER>
```

4.3.5.3 EXTERNAL STACK GROUP DECLARATION

```
129. <EXTERNAL STACK GROUP>!!=  EXTERNAL <ATTRIBUTE HEAD> <IDENTIFIER> \
                                <EXTERNAL STACK GROUP> , <IDENTIFIER>
```

4.3.5.4 EXTERNAL ARRAY DECLARATIONS

```
130. <ARRAY LIST>!!=  <IDENTIFIER> ( <EXTERNAL SIZE> ) \
                      <ARRAY LIST> , <IDENTIFIER> \
                      <ARRAY LIST> , <IDENTIFIER> ( <EXTERNAL SIZE> )
131. <EXTERNAL ARRAY DECLARATION>!!=  EXTERNAL <TYPE ATTRIBUTE> ARRAY
                                                          <ARRAY LIST> \
                                      EXTERNAL HALF ARRAY <ARRAY LIST>
```

4.3.6 STACK DECLARATIONS

```
132. <STACK NAME>!!= <IDENTIFIER> ( <INTEGER> )
133. <STACK DECLARATION>!!=  <TYPE ATTRIBUTE> STACK <STACK NAME> \
                             <FULL ATTRIBUTE> STACK <STACK NAME> \
                             <STACK DECLARATION> , <IDENTIFIER> \
                             <STACK DECLARATION> , <STACK NAME>
```

4.3.7 GLOBAL DECLARATIONS

```
134. <GLOBAL DECLARATION>!!=  GLOBAL <IDENTIFIER> \
                              <GLOBAL DECLARATION> , <IDENTIFIER>
```

4.3.8 FORMAT DECLARATIONS

```
135. <OPTIONAL $>!!=   $ \ <EMPTY>
136. <OPTIONAL ->!!=   - \ <EMPTY>
```

```
137. <FORMAT ELEMENT>!!=  <OPTIONAL $> <STRING> \
                          <INTEGER> ( <FORMAT LIST> ) \
                          R'<DECIMAL NUMBER>' \
                          <OPTIONAL $> P'<DECIMAL NUMBER>' F'<OPTIONAL ->
                                                 <DECIMAL NUMBER>' \
                          P'<DECIMAL NUMBER>' <OPTIONAL $> F'<OPTIONAL ->
                                                 <DECIMAL NUMBER>' \
                          <OPTIONAL $> F'<OPTIONAL -> <DECIMAL NUMBER>' \
                          <OPTIONAL $> I'<OPTIONAL -> <DECIMAL NUMBER>' \
                          <OPTIONAL $> O'<OPTIONAL -> <DECIMAL NUMBER>' \
                          <OPTIONAL $> H'<OPTIONAL -> <DECIMAL NUMBER>' \
                          <OPTIONAL $> D'<OPTIONAL -> <DECIMAL NUMBER>' \
                          <OPTIONAL $> S'<DECIMAL NUMBER>' \
                          <OPTIONAL $> E'<DECIMAL NUMBER>' \
                          <OPTIONAL $> L'<DECIMAL NUMBER>' \
                          <OPTIONAL $> A'<DECIMAL NUMBER>'
138. <FORMAT LIST>!!=  <FORMAT ELEMENT> \ <FORMAT LIST> , <FORMAT ELEMENT>
139. <FORMAT DECLARATION>!!=  FORMAT <IDENTIFIER> ( <FORMAT LIST> )
```

4.3.9 DEVICE DECLARATIONS

```
140. <DEVICE>!!=  CPRINT \ SPRINT \ MOF \ MTF \ KBOSS \ ICL
141. <DEVICE DECLARATION>!!=  DEVICE <IDENTIFIER> = <DEVICE> \
                              <DEVICE DECLARATION> , <IDENTIFIER>
                                                   = <DEVICE>
```

4.3.10 TASS DECLARATIONS

```
142. <TASS DECLARATION>!!=  TASS <TASS CONTROL CARDS> (SEE TEXT)
```

4.3.11 INSERT DECLARATIONS

```
143. <INSERT DECLARATION>!!=  INSERT <IDENTIFIER> ( <IDENTIFIER> )
```

4.3.12 COMPONENT DECLARATIONS

```
144. <COMPONENT ATTRIBUTE>!!=  <TYPE ATTRIBUTE> COMPONENT

145. <COMPONENT LIST>!!=  <COMPONENT ATTRIBUTE> <IDENTIFIER> \
                          <COMPONENT LIST> , <IDENTIFIER>

146. <COMPONENT OFFSET>!!=  OFFSET + <INTEGER> \ OFFSET - <INTEGER> \
                            OFFSET <INTEGER>

147. <OFFSET LIST>!!=  <COMPONENT OFFSET> FOR <IDENTIFIER> \
                       <OFFSET LIST> , <IDENTIFIER>

148. <SIGN EXTENSION>!!=  LOGICAL \ ARITHMETIC

149. <BIT FIELD>!!=  <SIGN EXTENSION> FIELD ( <INTEGER> , <INTEGER> ) \
                     FIELD ( <INTEGER> , <INTEGER> )

150. <FIELD LIST>!!= <BIT FIELD> FOR <IDENTIFIER> \
                     <FIELD LIST> , <IDENTIFIER>

151. <COMPONENT DECLARATION 1>!!=  <COMPONENT LIST> \ <OFFSET LIST> \
                                   <FIELD LIST>

152. <COMPONENT HEAD>!!=  <COMPONENT ATTRIBUTE> <IDENTIFIER>

153. <COMPONENT TAIL>!!=  ( <BIT FIELD> , <COMPONENT OFFSET> ) \
                          ( <COMPONENT OFFSET> , <BIT FIELD> ) \
                          ( <COMPONENT OFFSET> )

154. <COMPONENT DECLARATION 2>!!=  <COMPONENT HEAD> <COMPONENT TAIL>
```

4.3.13 COMMON DECLARATIONS

```
155. <COMMON HEAD>!!=  GLOBAL COMMON <IDENTIFIER> ;
                       EXTERNAL COMMON <IDENTIFIER> ; \

156. <COMMON ELEMENT>!!=  <TYPE DECLARATION> \ <ALPHA DECLARATION> \
                          <ARRAY DECLARATION> \ <STACK DECLARATION>

157. <COMMON LIST>!!=  <COMMON ELEMENT> ; \
                       <COMMON LIST> <COMMON ELEMENT> ;

158. <COMMON TAIL>!!=  ENDCOM \ COMEND

159. <COMMON DECLARATION>!!=  <COMMON HEAD> <COMMON LIST> <COMMON TAIL>
```

```
4.3.14  PRESET DECLARATIONS
        -------------------

160. <SIMPLE EXPRESSION LIST>!!=  <SIMPLE EXPRESSION> \
                                  <SIMPLE EXPRESSION LIST> ,
                                                   <SIMPLE EXPRESSION>

161. <PRESET ELEMENT>!!=  <VARIABLE> = <SIMPLE EXPRESSION LIST> \
                          <VARIABLE> TO <VARIABLE> =
                                                 <SIMPLE EXPRESSION LIST>

162. <PRESET LIST>!!=  <PRESET ELEMENT> \
                       <PRESET LIST> ; <PRESET ELEMENT>

163. <PRESET DECLARATION>!!=  PRESET <PRESET ELEMENT> \
                              PRESET BEGIN <PRESET LIST> END \
                              PRESET BEGIN <PRESET LIST> ; END

4.3.15  SYNONYM DECLARATIONS
        --------------------

164. <THLL ITEM>!!=  <CONSTANT> \ <IDENTIFIER> \
                     ANY OPERATOR OR DELIMITER

165. <THLL ITEM SEQUENCE>!!=  <EMPTY> \ <THLL ITEM> \
                              <THLL ITEM SEQUENCE> <THLL ITEM>

166. <SYNONYM RIGHTSIDE>!!=  <EMPTY> \ <CONSTANT> \
                             <ITEM DELIMETER> <THLL ITEM SEQUENCE>
                                                   <ITEM DELIMETER>

167. <ITEM DELIMETER>!!=  <THLL ITEM>  THAT IS NOT A <CONSTANT> OR
                     SEMICOLON AND DOES NOT OCCUR IN <THLL ITEM SEQUENCE>

168. <SYNONYM ELEMENT>!!=  <IDENTIFIER> = <SYNONYM RIGHTSIDE>

169. <SYNONYM LIST>!!=  <SYNONYM ELEMENT> \
                        <SYNONYM LIST> ; <SYNONYM ELEMENT>

170. <SYNONYM DECLARATION>!!=  SYNONYM <SYNONYM ELEMENT> \
                               SYNONYM BEGIN <SYNONYM LIST> END \
                               SYNONYM BEGIN <SYNONYM LIST> ; END
```

5. PROGRAMS

```
171. <PROGRAM HEAD>!!=  BEGIN <DECLARATION> \

                        <PROGRAM HEAD> ; <DECLARATION>

172. <PROGRAM>!!=  <IDENTIFIER> <PROGRAM HEAD> END FINIS \

                   <IDENTIFIER> <PROGRAM HEAD> ; END FINIS
```

APPENDIX B

TRICOMP COMPILER DIRECTIVES

B.1 GENERAL DESCRIPTION

The compiler directives are used to communicate various options to the compiler and to control the format of the printable listing.

The compiler enters the directive mode when either a \ or \\ is detected. The directive <u>mode</u> is terminated when the end of an input card has been detected. The \\ will allow the active card to appear on the printed listing. The \ will suppress printing of the active input card. The line counts will indicate that a line was suppressed. The \ and \\ will have no effect in a comment, TASS declaration, or SYNONYM definition. The \ or \\ can appear after any THLL item except FINIS and TASS;. If during a synonym expansion a \ or \\ occurs, it will cause the directive mode to be entered. The remainder of the synonym (even if on different cards in the synonym definition) and the remainder of the input card will be considered directives. A \ detected in a synonym expansion will cause that line to be suppressed. The following synonym definition will allow lines containing \ to be printed.

```
SYNONYM  \ = -\\- ;
```

In a similar manner, all directives may be suppressed in the listing.

More than one directive may be included on one input card. The directives have the following general form:

```
KEY   D   Y
```

where

KEY is a keyword identifier

D is an optional series (possibly empty) of THLL items that do not match the THLL item Y required by the KEY

Y is either an identifier, string, number, or signed number as required by the KEY. Some directives do not require a Y.

Examples:

```
(a)  \   LINE = 1

(b)  \   TITLE #'NAME OF MY TASK'

(c) \\   TITLE = #'',PAGE
```

The first example will cause the input lines to be double spaced. The second example will put a title on each page. The third example removes the user's title and causes a page to be ejected. Only the third example will be printed. The = and comma are completely optional.

The KEY used in the directives is just a predefined identifier which has no special meaning when not in the directive mode; therefore, the KEY words are not reserved words.

When the directive mode is entered, only a KEY or end of card will be recognized. After a key is recognized, only the Y or end of card will be recognized. If the Y is a string, processing continues until the string is terminated. Therefore, the user should be careful to terminate the string. Failure to do so can cause subsequent source cards to be incorporated in the string, since string processing does not terminate at the end of the card. The directive mode is restarted after a directive has been completely recognized.

Most of the options on the TRICOMP control card are available through directives. The directive KEY spelling, in some cases, will not be the same as the spelling of the option on the TRICOMP card. After the completion of a compile unit (program), the compiler will return to the state as defined on the TRICOMP card. Therefore, each compile unit can specify the resources needed for that program.

B.2 DIRECTIVE NOTATION

The following notation is used in the description of directives:

N - is an integer number. It may be a binary, octal, decimal or hexadecimal integer constant. The number must not be real or large enough to be considered double. An illegal number will cause that directive recognition to be aborted and the directive mode to be restarted. All signs are also ignored.

SN - is an integer number with optional sign. (See the description of N above.)

S - is a string. If the string is continued on the next card, the directive is ignored and directive processing is terminated.

ID - is an identifier. The identifiers used as KEYs may also be used without causing confusion. A new directive will not be started by using KEY as an identifier.

Except as noted, all directives are valid within one program. The compiler will return to the state specified on the TRICOMP control card after each program.

B.3 LISTING DIRECTIVES

1. PAGE - Immediately home paper. If this directive is printed, it would appear on the new page.

2. SPACE N - Print N blank lines before the next printable line. If the next printable line comes from an insert that is not being printed, then the directive is ignored. Spacing will not pass the end of page. Default is 0.

3. LINE N - Print N blank lines after each printable line. Spacing will not pass the end of page. The next printable line will appear at the top of the next page. Default is 0.

4. TAB N - Move the card image right to the N^{th} column. The integer number is limited to 48. Default is 0.

5. ATAB SN - Adjust the TAB number by SN. If SN is negative, the tab will move to the left. The tab number will not go below 0 or above 48. Tab numbers less than 0 are replaced by 0, but tab values greater than 48 are remembered but limited to 48.

6. TITLE S
   or
   TITLE -S - Print the string S at the top of each page just below the standard TRICOMP header. If the minus (-) is used before the string, then the string will begin in column 1 and its first character will be interpreted as a carriage control character. Otherwise, the string will be printed starting in column 21. This directive will be ignored if the string goes beyond the end of card. Long strings can be defined in a synonym and then expanded in the S position. A maximum of 130 characters including the carriage control character can be handled by the compiler. Longer strings will be truncated. If the standard TRICOMP header is suppressed, column 1 will be set to the home paper carriage control character and columns 121 through 130 will be used to specify the page number. An empty string will suppress the directive title line. Default is no title line.

7. NOHEADER - Suppress the standard TRICOMP header at the top of each page. If no directive title line is specified, only the home paper carriage control and page number will be printed for each new page. These will be added to the title line if it does exist, and the title line will be first on the page.

8. HEADER - Restore the standard TRICOMP header at the top of each page. If a directive title line was being printed, it must be re-defined because the home paper carriage control has been placed in that line. Both the standard TRICOMP header and title line would cause a new page to be ejected. HEADER is default.

9. RSIDE N - If N = 0, the right side of the listing is suppressed. If N = 1, restore the right side of the listing. This is helpful if a listing is to be printed on the intercom. The right side of the listing includes columns 73 through 90 of the input card image and the program name and line number. Default is 0.

10. MLINE N - Print a maximum of N lines to a page. Low value of N is limited to 10. There is no upper end of the value of N. Large values of N will have the effect of suppressing page headers. The standard TRICOMP header, directive title line (if it exists), and the top blank line are not counted towards this line count. Default is 56.

B.4 COMPILER DIRECTIVES

1. NOCODE - No code is to be generated. Pass 3 is suppressed. This is equivalent to P=2 on the TRICOMP control card or PASS=2 directive. Default is to include all passes if no fatal error occurs.

2. LIST N - This is the same as the L = octal number on the TRICOMP control card.

N = 0:	No listing except for errors and compile times.
N = 1:	List only the source input file, errors, cross-reference and compile times. This is default.
N = 2:	Also list the insert files in addition to the N = 1 option.

3. OPT N - Same as the OPT option on the TRICOMP control card. On the directive card, any THLL constant number can be used.

N = 0:	No optimization
N = 1:	Optimization is done.

Optimization is turned on by a single bit. Default is 1. This directive will cause the optimization to be altered at that point in the program.

4. BOUNDS N - This is the same as the B = octal number option on the TRICOMP control card.

N = 0: No runtime check of array subscripts will be performed.

N $\neq$ 0: Runtime array bounds check. This is default.

This directive will alter the bounds checking at that point in the program.

5. ABORT - This will cause the compiler to terminate the SCOPE job control stream if this or any of the remaining input programs has a fatal error. All of the input THLL programs will be compiled, but after the last program, TRICOMP will abort to an EXIT(S) card if any of the programs has a fatal error. EXIT and EXIT(U) control cards will not be honored when skipping SCOPE control cards. The directive will not cause an abort if only an earlier program has a fatal error. Default is to not abort.

6. SKIP - This turns on the skip mode of the compiler. All input between the SKPSTART and SKPEND directives will not be processed by the compiler. The SKPSTART directive must occur after the SKIP directive. Care must be taken because skipping can go beyond program boundaries.

7. NOSKIP - This turns off the compiler skip mode. The SKPSTART and SKPEND directives will not be honored. This is default.

8. SKPSTART - If the skip mode is active, then start skipping all input until a SKPEND directive. The SKPEND directive is the only directive that will be honored.

9. SKPEND - This terminates the skipping of the input text. It has no effect if there is not skipping. It can come from a synonym expansion. It cannot come from an insert file that is not already active. (Insert declarations are skipped.)

10. PRIV - This is the same as the EXEC option on the TRICOMP control card. This causes only the current program to be assembled in PRIV mode.

11. NOPRIV - This turns off the EXEC option on the TRICOMP control card for just the current program.

12. PASS N - This is the same as the P = octal number option on the TRICOMP control card.

N = 1, 2, 3, 4:	Execute N passes of the compiler, and then pass 5.
N = 5:	Execute all passes.

13. SCHEMA - This turns on the SCHEMA option on the TRICOMP control card for just the current program. Generate schema data.

14. NOSCHEMA - This turns off the SCHEMA option on the TRICOMP control card for just the current program. Do not generate schema data.

15. CRET - This turns on the CRET option on the TRICOMP control card for just the current program. Compile as a GDDF creating program.

16. NOCRET - This turns off the CRET option on the TRICOMP control card for just the current program. Compile as normal program. This is default unless CRET appeared on the TRICOMP control card.

17. GDDF - This turns on the GDDF option on the TRICOMP control card for just the current program. Compile as a program that is to be assembled using a GDDF.

18. NOGDDF - This turns off the GDDF option on the TRICOMP control card for just the current program. No GDDF is allowed in the assembly program generated by the compiler. This is default when GDDF is not specified on the TRICOMP control card.

19. CODEFILE ID - This is the same as the A = Filename option on the TRICOMP control card for just the current assembly. The default filename for the output to the assembler is BPCODE unless changed on the TRICOMP control card. Only the last filename specified by a CODEFILE directive will be used for that program. The code cannot be sent to more than one file.

20. XREF N - This is the same as the R = octal number option on the TRICOMP control card. This controls the level of cross-reference. This directive cannot be used after the first user-defined symbol is encountered.

21. T ID - This is the same as the T = ID option on the TRICOMP control card. The default filename for TASS control cards is TCARD unless changed on the TRICOMP control card. This directive will have effect only within one program. All the cards included in TASS declarations will go to the last specified file. Additional T directives will cause the cards in the following TASS declarations to go to the newly specified file. A T directive cannot be placed in a TASS declaration. The directive card would be written on the TASS control card file.

22. ICF N - This is the same as the ICF = octal number option on the TRICOMP control card for only the current program.

23. DSS N - This is the same as the DSS = octal number option on the TRICOMP control card for only the current program.

24. RSS N - This is the same as the RSS = octal number option on the TRICOMP control card for only the current program.

25. LAB N - This is the same as the LAB = octal number option on the TRICOMP control card for only the current program.

26. HEAD N - This is the same as the HEAD = octal number option on the TRICOMP control card for only the current program.

27. FSL N - This is the same as the FSL = octal number option on the TRICOMP control card for only the current program. In order for this directive to have an effect, it must occur before the first format declaration.

28. SYN N - This is the same as the SYN = octal number option on the TRICOMP control card for only the current program. In order for this directive to have an effect, it must occur before the first synonym definition.

29. DEBUG N - Compiler systems use.

30. CGOPTS N - Compiler systems use.

31. BIN S - This is the same as the BIN = PPNNNNN option on the TRICOMP control card. It applies to current compile unit. When the compile unit has been completely processed, the compiler reverts to the mode specified on the TRICOMP card. The string S must be of the form such that it represents the name of a Binder Library, for example BIN = #/BTLIBR/. If the string contains more than eight characters, it will be truncated to eight characters. The use of a null string, such as BIN = #//, will cause the compiler to generate source code for TASS on the file specified by the A option on the TRICOMP card, the file specified by the CODEFILE compiler directive, or the default file BPCODE.

32. MAXBLK N - This is the same as the MAXBLK = octal number option on the TRICOMP control card for only the current program. In order for this directive to have an effect, it must occur before the first user-defined block. Only the first occurrence of this directive in a program will be honored.

33. IDSS N - This is the same as the IDSS = octal number option on the TRICOMP control card for only the current program. In order for this directive to have an effect, it must occur before the first user-defined block is opened. Only the first occurrence of this directive in a program will be honored.

34. SCANMAX N - This is the same as the SCANMAX = octal number option on the TRICOMP control card for only the current program.

35. ICFMIN N - This is the same as the ICFMIN = octal number option on the TRICOMP control card for only the current program.

36. ICFPAGE N - This is the same as the ICFPAGE = octal number option on the TRICOMP control card for only the current program.

37. TMPMAX N - This is the same as the TMPMAX = octal number option on the TRICOMP control card for only the current program.

38. OWNBR N - This is the same as the OWNBR = octal number option on the TRICOMP control card for only the current program.

39. INSBR N - This is the same as the INSBR = octal number option on the TRICOMP control card for only the current program.

40. CONBR N - This is the same as the CONBR = octal number option on the TRICOMP control card for only the current program.

APPENDIX C

SYNTAX DIAGRAM CROSS-REFERENCE

	DEFINED	USED
constant	6	21, 54
CPRINT		47
D		9, 46
data declaration	37	29
decimal digit	8	9, 13
decimal integer	9	10, 11, 12, 39, 46
decimal number	11	7
declaration	29	55
DEFINE		32
delimiting character		13
descriptor head	33	32
DEVICE		30, 42, 47
device	47	47
device declaration	47	37
device id		15
digit		3
DO		24, 28
DOUBLE		29
E		9, 12, 46
ELSE		23
END		27, 54, 55
ENDCASE		24
ENDCOM		52
ENDIF		23
ENTRYP		15
EQL		20
EXEC		34
executive head	34	33
EXIT		25
exit statement	25	25
expression	14	2, 15, 16, 21, 23, 24, 25, 26, 27, 28
EXTERNAL		41, 42, 43, 52
external size	31	35, 36, 43
external array declaration	43	41
external declaration	41	37
external procedure declaration	41	41
external stack group	42	41
external variable declaration	42	41
F		9, 46
FALSE		12
FIELD		51
field list	50	49
FINIS		55
FOR		16, 28, 50, 51
FORMAT		30, 42, 45
format declaration	45	37

	DEFINED	USED
format element	46	45
format id		15, 21
format list	45	45, 46
full attribute	29	38, 39, 44
function designator	15	21
GEQ		20
GLOBAL		1, 44, 52
global declaration	44	37
GOTO		25
goto statement	25	25
GRT		20
H		46
HALF		36, 38, 43
hex digit	9	11
hex number	11	7
I		46
ICL		47
identifier	13	1, 13, 33, 34, 35, 38, 39, 40, 42, 43, 44, 45, 48, 49, 50, 51, 52, 54, 55
IF		23
IFEND		23
INSERT		48
insert declaration	48	37
INTEGER		29
integer	7	6, 31, 34, 44, 46, 50, 51
INTERRUPT		34
item delimiter	54	54
K		10
KBDSS		47
L		46
label id		25, 27, 40
label list	40	40
LEQ		20
LES		20
letter	13	3, 13
LINK		30
LOC		21
LOGICAL		51
loop argument	16	15
LOOPEXIT		25
loop statement	28	26
MDF		47
MTF		47
NEQ		20
NOT		20
NOTE		22

	DEFINED	USED
NULL		28
null statement	28	26
number	6	6
O		46
octal digit	8	8, 11
octal number	11	7
OFFSET		50, 51
offset list	51	49
OPTARG		33
OR		20
ORB		22
OWN		29, 38, 40
P		46
POINTER		29
preset declaration	53	37
preset element	53	53
primary	21	22
PROCEDURE		30, 33
procedure attribute	30	33, 35, 36
procedure body	32	31
procedure declaration	31	29
procedure head	32	31
procedure head 1	33	33
procedure head 2	33	32, 33
procedure head 3	33	32
procedure id		15, 21
procedure specification	35	34, 41
program	55	
program head	55	27, 55
proper statement	26	24, 28
R		46
REAL		29
real number	12	6, 12
relational operator	20	20
REPEAT		16, 28
RETURN		2, 26
return statement	26	25
S		46
scaled real number	12	6
scale part	10	10, 11, 12
simple expression	20	14
simple expression list	53	53
special character	3	3
specification element	34	34
specification part	34	32
SPRINT		47
STACK		42, 44

	DEFINED	USED
stack declaration	44	37, 52
stack id		14, 15
statement	24	23, 24, 27
STEP		16, 28
string	13	6
SWITCH		40
switch declaration	40	37
switch id		25
SYNONYM		54
synonym declaration	54	37
synonym element	54	54
synonym rightside	54	54
TASS		48
tass control cards		48
tass declaration	48	37
THEN		23
THLL item	54	54
TRUE		1, 12
type attribute	29	29, 30, 34, 35, 36, 38, 39, 42, 43, 44, 49, 52
type declaration	39	37, 52
UNTIL		16, 28
VALUE		30, 33
value attribute	30	36
variable	14	21, 24, 28
variable id		14, 16
WHILE		16, 28
X		11
XOR		20
XORB		22
Ø		7
1		7
2		8
3		8
4		8
5		8
6		8
7		8
8		8
9		8
+		10, 12, 22, 50, 51
-		10, 11, 12, 22, 46, 50, 51
'		10, 11, 46
.		12, 13
#		13

	DEFINED	USED
(		14, 15, 21, 25, 33, 35, 36, 38, 39, 43, 44, 45, 46, 48, 50, 51
)		14, 15, 21, 25, 33, 35, 36, 38, 39, 43, 44, 45, 46, 48, 50, 51
,		14, 15, 16, 23, 24, 31, 33, 35, 38, 39, 40, 42, 43, 44, 45, 47
=		16, 24, 28, 40, 47, 54
{		16
}		16
*		22, 31
/		22
**		22
;		27, 31, 32, 33, 52, 54, 55
:		27
$		46

DISTRIBUTION

Defense Technical Information Center Cameron Station Alexandria, VA 22314	(12)
Library of Congress Washington, DC 20540 ATTN: Gift and Exchange Division	(4)
GIDEP Operations Office Corona, CA 91720	

Local

K50-GE	
K50-EG&G (Library)	
K51	
K53	(2)
K54	(6)
K55	(2)
K56	(2)
K70	
X210	(6)
X211	(2)
E41	

LMED

-8