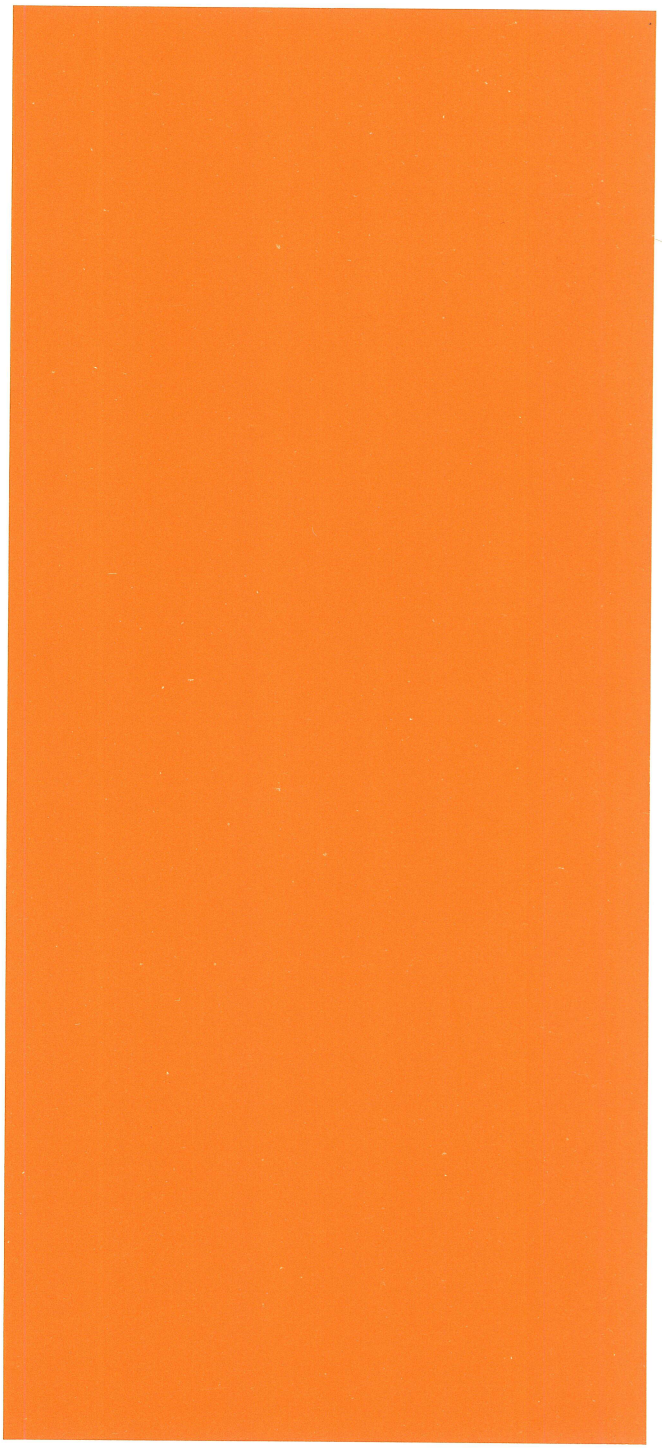
Honeywell Bull

ALGOL

SERIES 600/6000

SOFTWARE



BS 11, Rev. 0

Ref.: 19.30.220 A

Honeywell Bull ALGOL

SERIES 600/6000

SUBJECT:

Description and Use of The ALGOL Language.

SPECIAL INSTRUCTIONS:

This manual, Order Number BS11, Rev. 0, supersedes CPB-1657, dated March 1970 and its Addendum No. 1, dated October 1971. The new order number is assigned to be consistent with the overall Honeywell publications numbering system. Additions and changes from the previous editions are indicated by change bars in the margins.

SOFTWARE SUPPORTED:

Series 600 Software Release 5.0
Series 6000 Software Release C

DATE:

June 1972

ORDER NUMBER:

BS11, Rev. 0 (Formerly CPB-1657)

Printed in France Ref.: 19.30.220 A

PREFACE

This manual is intended as a reference for the programmer of the Series 600/6000 ALGOL language. The language is defined and instructions are given as to methods of writing ALGOL programs for both batch and time-sharing environments.

 File No.: 1623,1723

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 600 SYSTEM

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 600:		
Hardware reference:			
Series 600	System Manual	371	BM78
DATANET 355	DATANET 355 Systems Manual	1645	BS03
Operating system:			
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	1518	BR43
Control Card Formats	Control Cards Reference Manual	1688	BS19
System initialization:			
GCOS Startup	System Operating Techniques	DA10	DA10
Communications System	GRTS/355 Startup Procedures Reference Manual	1715	BJ70
Storage Subsystem Startup	DSS180 Disk Storage Subsystem Startup Procedures	DA11	DA11
Data management:			
File System	GCOS File System	1513	BR38
Integrated Data Store (I-D-S)	Integrated Data Store	1565	BR69
File Processing	Indexed Sequential Processor	DA37	DA37
Multi-Access I-D-S	Multi-Access I-D-S Implementation Guide	DA80	DA80
File Input/Output	General File and Record Control System	1003	BN85
Program maintenance:			
Object Program	Source and Object Library Editor	1723	BJ71
System Editing	System Library Editor	1687	BS18
Test system:			
On-Line Peripheral testing	GCOS On-Line Peripheral Test System (OPTS-600)	1573	BR76
Total On-Line testing	Total On-Line Test System (TOLTS)	DA49	DA49
Language processors:			
Macro Assembly Language	Macro Assembler Program	1004	BN86
COBOL Language	COBOL Compiler	1652	BS08
COBOL Usage	COBOL User's Guide	1653	BS09
ALGOL Language	ALGOL	1657	BS11
JOVIAL Language	JOVIAL	1650	BS06
FORTRAN Language	FORTRAN	1686	BJ67
FORTRAN IV Language	FORTRAN IV	1006	BN88
DATANET 355	DATANET 355 Macro-Assembler Program	1660	BB98
Generators:			
Sorting	Sort/Merge Program	1005	BN87
Merging	Sort/Merge Program	1005	BN87
Simulators:			
DATANET 355 Simulation	DATANET 355 Simulator Reference Manual	1663	BW23

BS11

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
Series 600:			
Remote terminal system:			
DATANET 30	NPS/30 Programming Reference Manual	1558	BR68
DATANET 30/305/355	GRTS Programming Reference	DA79	DA79
Service and utility routines:			
Loader	General Loader	1008	BN90
Utility Programs	Utility	1422	BQ66
Conversion	Bulk Media Conversion	1096	BP30
System Accounting	GCOS Accounting Summary Edit Programs	1651	BS07
FORTRAN	FORTRAN Subroutine Libraries Reference Manual	1620	BR95
Controller Loader	Relocatable Loader	DA12	DA12
Service Routines	Service Routines	DA97	DA97
Time-sharing systems:			
Operating System	GCOS Time-Sharing System General Information	1643	BS01
System Programming	GCOS Time-Sharing Terminal/Batch Interface Facility	1642	BR99
System Programming	GCOS Time-Sharing System Programmers' Reference Manual	1514	BR39
BASIC Language	Time-Sharing BASIC	1510	BR36
FORTRAN Language	Time-Sharing FORTRAN	1566	BR70
Text Editing	Time-Sharing Text Editor	1515	BR40
Transaction processing:			
User's Procedures	Transaction Processing System User's Guide	DA82	DA82
Site Operations	Transaction Processing System Site Manual	DA13	DA13
Handbooks:			
Console Messages	Console Typewriter Messages	1477	BR09
Index	Comprehensive Index	1499	BR28
Pocket guides:			
Time-Sharing Programming	GCOS Time-Sharing System	1661	BS12
Macro Assembly Language	GMAP	1673	BS16
COBOL Language	COBOL	1689	BJ68
Control Card Formats	GCOS Control Cards & Abort Codes	1691	BJ69
Software maintenance (SMD):			
Table Definitions	GCOS Introduction & System Tables SMD	1488	BR17
Startup program	Startup (INIT) SMD	1489	BR18
Input System	System Input SMD	1490	BR19
Peripheral Allocation	Dispatcher and Peripheral Allocation SMD	1491	BR20
Core Allocation/Rollcall	Rollcall, Core Allocation and Operator Interface SMD	1492	BR21
Fault Processing	Fault Processing SMD	1493	BR22
Channel Modules	I/O Supervisor (IOS) SMD	1494	BR23
Error Processing	GCOS Exception Processing SMD	1495	BR24
Output System	Termination and System Output SMD	1496	BR25
File System Modules	File System Maintenance SMD	1497	BR26
Utility Programs	GCOS Utility Routines SMD	1498	BR27
Time-Sharing System	Time-Sharing Executive SMD	1501	BR29

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 6000 SYSTEM

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
Series 6000:			
Hardware reference:			
Series 6000	Series 6000 Summary Description	DA48	DA48
DATANET 355	DATANET 355 Systems Manual	1645	BS03
Operating system:			
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	1518	BR43
Control Card Formats	Control Cards Reference Manual	1688	BS19
System initialization:			
GCOS Startup	System Startup and Operation	DA06	DA06
Communications System	GRTS/355 Startup Procedures Reference Manual	1715	BJ70
Storage Subsystem Startup	DSS180 Disk Storage Subsystem Startup Procedures	DA11	DA11
Data management:			
File System	GCOS File System	1513	BR38
Integrated Data Store (I-D-S)	Integrated Data Store	1565	BR69
File Processing	Indexed Sequential Processor	DA37	DA37
Multi-Access I-D-S	Multi-Access I-D-S Implementation Guide	DA80	DA80
File Input/Output	General File and Record Control System	1003	BN85
Program maintenance:			
Object Program	Source and Object Library Editor	1723	BJ71
System Editing	System Library Editor	1687	BS18
Test system:			
On-Line Peripheral Testing	GCOS On-Line Peripheral Test System (OPTS-600)	1573	BR76
Total On-Line testing	Total On-Line Test System (TOLTS)	DA49	DA49
Language processors:			
Macro Assembly Language	Macro Assembler Program	1004	BN86
COBOL Language	COBOL Compiler	1652	BS08
COBOL Usage	COBOL User's Guide	1653	BS09
ALGOL Language	ALGOL	1657	BS11
JOVIAL Language	JOVIAL	1650	BS06
FORTRAN Language	FORTRAN	1686	BJ67
DATANET 355	DATANET 355 Macro-Assembler Program	1660	BB98
Generators:			
Sorting	Sort/Merge Program	1005	BN87
Merging	Sort/Merge Program	1005	BN87
Simulators:			
DATANET 355 Simulation	DATANET 355 Simulator Reference Manual	1663	BW23

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 6000:		
Service and utility routines:			
Loader	General Loader	1008	BN90
Utility Programs	Utility	1422	BQ66
Conversion	Bulk Media Conversion	1096	BP30
System Accounting	GCOS Accounting Summary Edit Programs	1651	BS07
FORTRAN	FORTRAN Subroutine Libraries Reference Manual	1620	BP95
Controller Loader	Relocatable Loader	DA12	DA12
Service Routines	Service Routines	DA97	DA97
Time-sharing systems:			
Operating System	GCOS Time-Sharing System General Information	1643	BS01
System Programming	GCOS Time-Sharing Terminal/Batch Interface Facility	1642	BR99
System Programming	GCOS Time-Sharing System Programmers' Reference Manual	1514	BR39
BASIC Language	Time-Sharing BASIC	1510	BR36
FORTRAN Language	FORTRAN	1686	BJ67
Text Editing	Time-Sharing Text Editor	1515	BR40
Remote terminal system:			
DATANET 30	NPS/30 Programming Reference	1558	BR68
DATANET 30/305/355	GRTS Programming Reference	DA79	DA79
Transaction processing:			
User's Procedures	Transaction Processing System User's Guide	DA82	DA82
Site Operations	Transaction Processing System Site Manual	DA13	DA13
Handbooks:			
Console Messages	Console Typewriter Messages	1477	BR09
Index	Comprehensive Index	1499	BR28
Pocket guides:			
Time-Sharing Programming	GCOS Time-Sharing System	1661	BS12
Macro Assembly Language	GMAP	1673	BS16
COBOL Language	COBOL	1689	BJ68
Control Card Formats	GCOS Control Cards and Abort Codes	1691	BJ69

Rev. 7207

CONTENTS

ILLUSTRATION

TABLE

SECTION I

INTRODUCTION

DEFINITION OF ALGOL

ALGOL (ALGOrithmic Language) is a computer language with the unique capability for expressing problem solutions as efficient and precise procedures.

The ALGOL language can be considered an international algorithmic language for computing, for it appears to be universal in concept and is effective for stating a wide class of algorithms for numerical mathematics and for some logical processes.

ALGOL is comprised of a set of symbols and a set of rules. Associated with these are a set of definitions which are peculiar to a description of the language, its form, and use.

STRUCTURE OF ALGOL

ALGOL is composed of statements and declarations.

- Statements are used to specify operations to be performed by the computer in solving a problem.
- Declarations provide the ALGOL compiler with information needed to define and link together various elements of the computer program during processing. In addition, the existence of declarations within the language facilitates the definition of program parameters.

The statements and declarations in turn are composed of symbols. Some ALGOL symbols might conventionally be termed "character strings"; however, the definition of a symbol in ALGOL does not imply a single character. Certain symbols are enclosed in apostrophes. These apostrophes are a part of the symbol and must always appear when the symbol is used.

Statements and declarations are translated by the ALGOL compiler and the program is executed under the control of the Comprehensive Operating Supervisor (GCOS). Only those rules and symbols which govern ALGOL programming on a Series 600/6000 computer system are considered in this manual.

BASIC SYMBOLS

The basic symbols utilized for writing ALGOL programs are as follows:

- uppercase letters A to Z
- digits 0 to 9
- logical values - 'TRUE', 'FALSE'
- special symbols

OPERATOR SYMBOLS

Arithmetic Operators

Symbol	Definition[1]
+	addition
-	subtraction
*	multiplication
/	division
%	division
↑	exponentiation

Relational Operators

Symbol	Definition
'LS'	less than (<)
'LQ'	less than or equal to (≤)
'EQ'	equal to (=)
'GQ'	greater than or equal to (≥)
'GR'	greater than (>)
'NQ'	not equal to (≠)

Logical Operators

Symbol	Definition
'EQV'	equivalent (≡)
'IMP'	implies (⊃)
'OR'	or (∧)
'AND'	and (∨)
'NOT'	negation (¬)

---

[1] The symbols shown in this column are not necessarily available to the user for coding. They are included to show the mathematical meaning of the corresponding ALGOL symbol.

PUNCTUATION SYMBOLS

The following symbols have definite functions in the ALGOL language.

Symbol	Definition	Use
.	period	decimal point in numbers
,	comma	separator for items in a list
:	colon	separator for statement label
;	semicolon	separator for statements
(	left parenthesis	enclose parameter lists; indicate
)	right parenthesis	expression evaluation
[	left bracket	enclose subscripts
]	right bracket	enclose subscripts
"	left string quote	enclose strings
\	right string quote	enclose strings
'	apostrophe	indicate exponent
←	arrow	assignment operator
	blank space	space within strings

RESERVED WORDS

These special symbols have a fixed meaning in the ALGOL language.

'ARRAY'	'LINK'
'BEGIN'	'NONLOCAL'
'BOOLEAN'	'OWN'
'CODE'	'PROCEDURE'
'COMMENT'	'REAL'
'DO'	'RENAME'
'ELSE'	'STEP'
'END'	'STRING'
'EXTENDED REAL'	'SWITCH'
'FOR'	'THEN'
'GOTO'	'UNTIL'
'IF'	'VALUE'
'INTEGER'	'WHILE'
'LABEL'	

STATEMENT TYPES

There are six types of statements available in ALGOL. Their names and a brief description of their functions follow:

Statement Types

Name	Function
Assignment	To perform calculations and to assign a value to a variable or a group of variables
Conditional	To control the execution of individual statements or groups of statements
Dummy	To satisfy a programming protocol but it in itself performs no operation

'FOR'	To iterate a sequence of statements
'GOTO'	To transfer control
Procedure	To call a previously defined sequence of statements (e.g., a subroutine)

DECLARATION TYPES

There are five types of declarations available in ALGOL. Their names and a brief description of their functions follow:

Declaration Types

Name	Function
'ARRAY'	To define an array, specify its dimensions and its type
'PROCEDURE'	To define a subset of the computer program (e.g., a subroutine)
'SWITCH'	To specify control parameters which govern the sequence of program execution
Type	To specify the kind of value which a variable is to represent
'LINK'	To permit separate compilation of parts of a program

COMBINING STATEMENTS

The ALGOL language is structured in such a way as to impose rules of combining statements and segregating these as programs or subprograms in their own right. These concepts are presented in "Compound Statement and Block", Section VII.

INPUT/OUTPUT PROCESS

ALGOL does not contain statements which allow direct control of the input/output process. Thus, no statements or declarations exist for reading from or writing on external devices (e.g., READ, WRITE, etc., as in FORTRAN). To accomplish the usual input/output operations, procedures are provided which may be "called" by the user as subroutines. These procedures are described in "Input/Output", Section VIII.

SECTION II

WRITING AN ALGOL PROGRAM

FORM OF AN ALGOL PROGRAM

ALGOL programs are divided into logical sections called blocks. The entire program is also a block and must be enclosed within the symbols 'BEGIN' and 'END'. A block may contain any number of sub-blocks within it.

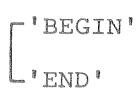
Variables, arrays, procedures and switches which are used in a block are defined in declarations at the beginning of the block. These declarations are followed by the statements of the block. Any statement of a block may in itself be a block (i.e., it must have block format as described in Section VII) and thus blocks may be nested to any depth.

All ALGOL statements may be labelled with one or more statement labels; i.e., simple statements, compound statements, and blocks may be labelled.

Execution of an ALGOL program starts with the first statement and continues successively from statement to statement. However, certain statements in the language have the power to change the sequence of statement execution.

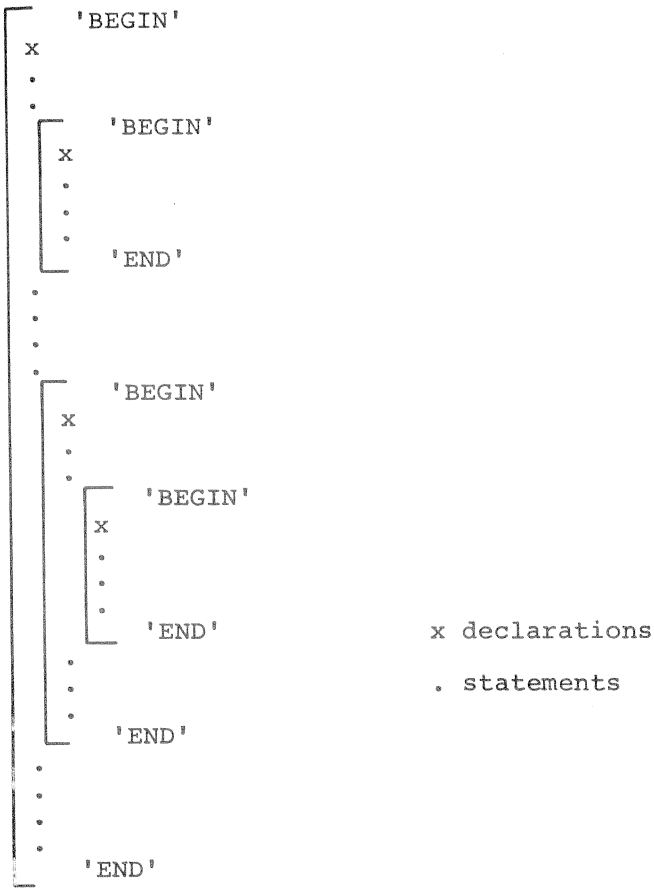
Execution of the program is terminated when control reaches the 'END' symbol of the outermost block of the program.

The following diagram portrays visually the structure of a typical (though arbitrary) ALGOL program. Each bracket denoted by



represents a block.

The blocks are composed of declarations and statements (as discussed above). The declarations must precede the statements. This diagram represents an ALGOL program with three block levels and four blocks.



WRITING RULES AND TECHNIQUES

Coding

The ALGOL program may be written on any 72-column coding form but writing will be facilitated if the program is written on forms designed specifically for the language.

Columns 1-72 of the coding form may be used for ALGOL statements and declarations. ALGOL code may appear anywhere within these columns.

The coding may be free-form. That is, any number of statements and/or declarations may appear on a single line.

A single statement or declaration may occupy as many lines as is required.

Blanks may be used freely throughout the ALGOL code to improve the readability of the text. The only place in ALGOL in which blanks are significant is in strings. In all other instances they are disregarded by the compiler.

Since the line format of ALGOL programs is very flexible, it is suggested that statement levels be indented on a new line to improve ease of reading and understanding a program.

Thus each new 'BEGIN' symbol may be indented at a new margin, and the 'END' corresponding to the 'BEGIN' may be placed at the same margin. Since statements may contain other statements, each lower statement level may be indented. When a higher level is resumed, statements for that level may be placed at the proper level margin (see form of the sample program given at the end of this Section).

Line indenting will in no way affect program execution but tends to make the program structure easier to follow.

Punctuation

When writing ALGOL statements and declarations, there are two important rules of punctuation which must be observed.

1. The semicolon is used as the symbol between statements and between declarations. However, the semicolon may be omitted after the last simple statement of a compound statement or block. The symbol 'END' serves as a statement separator in this case.

   Examples:

   1. A ← 2; 'GOTO' Z

   2. 'BEGIN' 'INTEGER' A; 'REAL' B; A ← 5.3; B ← 7.2 'END'

2. The colon is used as the symbol to separate a statement label from a statement.

   Examples:

   1. L: A ← B+C; P: 'GOTO' R

   2. T: 'BEGIN' I ← I+1; J ← J+1 'END'

Insertion of Comments

If it is desired to place comments within the text of an ALGOL program, it may be done as follows:

1. To insert a comment between statements or declarations, or at the beginning of a compound statement or a block, the comment must be enclosed within the symbols 'COMMENT' and semicolon.

   Examples:

   1. A ← B; 'COMMENT' COMPUTING C; C ← A

   2. 'BEGIN' 'COMMENT' COMPUTING C; A ← B; C ← A 'END'

2. To place a comment after a compound statement or a block (i.e., after the symbol 'END') the symbol 'COMMENT' is not necessary. A semicolon must be used after the text if an 'END' or 'ELSE' symbol does not appear.

Examples:

1. 'BEGIN' A ← B; C ← A 'END' COMPUTING D; D ← C

2. 'IF' A 'LS' B 'THEN' 'BEGIN' A ← B;
C ← A 'END' COMPUTING C D IF A LS B 'ELSE' B ← A

A SAMPLE ALGOL PROGRAM

To illustrate the fundamentals of ALGOL, a sample ALGOL program is provided.

The purpose of this sample program is to merge two sets of numbers. The two sets are contained in locations a(1), a(2),...,a(i),..., a(n) and b(1), b(2),...,b(j),...,b(m). The numbers in each set are assumed to be arranged in increasing order. The merged set is contained in locations c(1), c(2),..., c(k),... .

The program operates as follows. The elements of arrays a and b are compared. At each comparison, the smaller element is put into the result array c. When the end of either array a or b is reached, any remaining elements in the other array are put into the result array.

Symbols used in the program:

Symbol	Description
A	Identifier of input array
B	Identifier of input array
C	Identifier of output array
N	Subscript bound for array A
M	Subscript bound for array B
R	Subscript bound for array C
I	Subscript for array A
J	Subscript for array B
K	Subscript for array C
P	Controlled variable of 'FOR' statement

A listing of the program follows. The program is assumed to be a block contained in a larger block wherein the value of N, M, and R are assigned, and wherein P is defined.

```
                    Program                                          Line

'BEGIN' 'ARRAY' A[1:N], B[1:M], C[1:R]; 'INTEGER' I,                 100
    J, K; I←J←K←1;                                                   110

    START: 'IF' I 'GR' N 'THEN'                                      120

            'BEGIN' P←0; Q: C[K+P]←B[J+P]; P←P+1;                    130
                'IF' P 'LQ' M-J 'THEN' 'GOTO' Q 'END'

        'ELSE' 'IF' J 'GR' M 'THEN'                                  140

            'BEGIN' P←0; S: C[K+P]←A[I+P]; P←P+1;                    150
                'IF' P 'LQ' N-I 'THEN' 'GOTO' S 'END'

                    'ELSE' 'BEGIN'                                   160

                        'IF' A[I] 'GQ' B[J] 'THEN'                   170

                            'BEGIN' C[K]←B[J];                       180
                                J←J+1 'END'

                        'ELSE' 'BEGIN' C[K]←A[I];                    190
                            I←I+1 'END';

                        K←K+1; 'GOTO' START                          200

                        'END'                                        210

'END'                                                                220
```

A line by line description of this program is given below.

Line	Description
100	Contains 'BEGIN' for the block, and declarations of variables used.
110	Contains an assignment statement which sets I, J, and K to the value 1.
120	Contains statement label "START" and the beginning of a conditional statement which extends to line 210. The 'IF' clause checks whether all of the elements of array A have been compared.
130	Contains the true branch of the 'IF' clause of line 120. The true branch is a compound statement which moves the remaining elements, if any, of array B to array C.
140	Contains the start of the false branch of the 'IF' clause of line 120. The false branch extends to line 210. The 'IF' clause in this line checks whether all of the elements of array B have been compared.
150	Contains the true branch of the 'IF' clause of line 140. The true branch is a compound statement which moves the remaining elements of array A to array C.
160	Contains the start of the false branch of the 'IF' clause of line 140. The false branch is a compound statement enclosed within 'BEGIN' and 'END' and extends to line 210.

170 Contains an 'IF' clause which compares elements of arrays A and B.

180 Contains the true branch of the 'IF' clause of line 170. The true branch is a compound statement which moves an element of array B to array C and then updates the B array subscript, J.

190 Contains the false branch of the 'IF' clause of line 170. The false branch is a compound statement which moves an element of array A to array C and then updates the A array subscript, I.

200 Contains an assignment statement to update the C array subscript, K, and a 'GOTO' statement to transfer control to the statement labelled "START."

210 Contains 'END' for the compound statement starting on line 160.

220 Contains 'END' for the block starting on line 100.

The structure of the conditional statement of the program is shown in Figure 2-1.

If the condition of line 120 is true, the true branch, line 130, is taken and subsequent control goes to line 210; i.e., the false branch is skipped. If the condition of line 120 is false, control goes to the false branch, line 140.

If the condition of line 140 is true, the true branch, line 150, is taken and subsequent control goes to line 210; i.e., the false branch is skipped. If the condition of line 140 is false, control goes to the false branch, line 160.

If the condition of line 170 is true, the true branch, line 180, is taken and subsequent control goes to line 210; i.e., the false branch is skipped. If the condition of line 170 is false, control goes to the false branch, line 190.

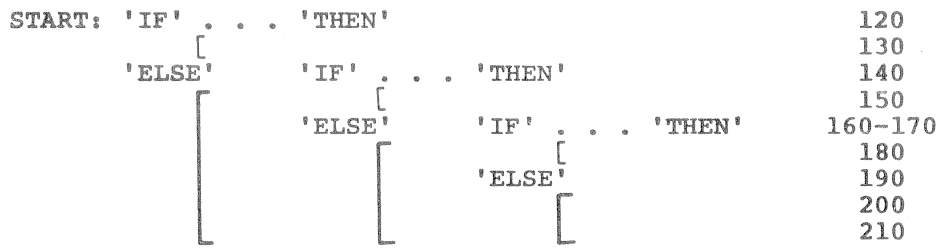


Figure 2-1. Outline of Conditional Statement

SECTION III

ALGOL LANGUAGE DEFINITIONS

The following terms are defined as they apply to the ALGOL language.

IDENTIFIER

A name given to a variable, an array, a label, a switch, and a procedure. The name may be composed of any number of letters and digits. However, the name must begin with a letter.

Example:

A, BETA, M12, T12C7, TOTALAMOUNT

Blanks are not considered significant in ALGOL except in strings, and they may be used freely within identifiers.

Example:

AB LE is considered the same identifier as ABLE or ABL E.

Two different quantities may not have the same identifier unless they appear in different blocks. (See "Compound Statement and Block", Section VII for clarification.) Certain identifiers are recognized as standard procedures by the ALGOL compiler. (See the list of reserved identifiers in Appendix A).

NUMBER

Integer, real number, extended real number.

INTEGER

A whole number written without a decimal point consisting of 1 to 11 decimal digits. The range of an integer n is:

$$-2^{35} \leq n \leq 2^{35} - 1$$

The precision is to 10 decimal digits.

Positive integers may have no sign. Negative integers must be preceded by a minus sign.

Examples:

0,-452,+7586421,33

REAL NUMBER

A series of from 1 to 9 decimal digits written with or without a decimal point. If the decimal point appears, it must not be the last character.

An exponent part may be added to specify the integral power of 10 to which the number must be raised. The exponent part is separated from the digits by an apostrophe ('). The exponent may also appear alone.

The range of a real number n is:

$$-2^{127} \leq n \leq 2^{127}$$

The precision is to 8 decimal digits. Positive real numbers do not require a sign. However, a plus sign is permitted. Negative real numbers require a minus sign.

Examples:

15.7, -.0045, +25.0, 1.7'-3, 5'3

EXTENDED REAL NUMBER

A series of from 1 to 19 decimal digits written with or without a decimal point. If the decimal point appears, it may not be the last character.

An exponent part may be added to specify the integral power of 10 to which the number must be raised.

The exponent part may also appear alone.

The range of an extended real number is:

$$-2^{127} \leq n \leq 2^{127}$$

The precision is to 18 decimal digits.

Positive extended real numbers do not require a sign. However a plus sign is permitted. Negative extended real numbers require a minus sign.

Examples:

135982.7834, 21.762'-19

STRING

A sequence of basic symbols enclosed in the left and right string quotes (" and \ ); or a sequence of basic symbols and strings enclosed in the string quotes.

Strings may be used as actual parameters of procedures.

Examples of strings:

"A B C \

" A B " CDE\FG\

VARIABLE

A quantity referred to by a name (the variable identifier) whose value may be changed.

The kind of quantity a variable may represent is determined by a type declaration and may be either integer, real, extended real, or Boolean.

Example:

X, ABC, YZ5N,

THIS IS A VARIABLE

SUBSCRIPTED VARIABLE

A subscripted variable has the form $a[b_1,b_2,...,b_n]$ where a is an array identifier and $b_1,b_2,...,b_n$ are arithmetic expressions.

The number of subscripts n must be the same as the number of dimensions declared for a.

Each subscript b_i acts like a variable of type 'INTEGER' and the evaluation of the subscript is understood to be equivalent to an assignment to this integer variable.

Evaluation of subscripts within a subscript list proceeds from left to right. The value of the subscripted variable is defined only if the value of each subscript expression is within the subscript bounds of the array.

Example:

AB[1,3], BOY['IF' B 'EQ' C 'THEN' 1 'ELSE' 2]

SIMPLE ARITHMETIC EXPRESSION

A sequence of numbers, variables, subscripted variables, or function calls separated by arithmetic operators and parentheses, which represents a rule for computing a numerical value.

1. All quantities used in an arithmetic expression must be of type real, extended real, or integer.

2. For operators +, -, or *, the result of calculation will be integer if both operands are integer; real if both operands are real; and extended real for all other cases.

3. There are two operators which denote division-- / and %. Both are defined for all combinations of real, extended real, and integer quantities; however,

   a. / will give a result of type real only if both operands are real. In all other cases, the result will be of type extended real.

   b. % will give the same results as / except that the value will always be integral. The result is truncated not rounded to an integer; i.e., 5 % 3 = 1.

4. Exponentiation -

   a. a↑b↑c is equivalent to $(a^b)^c$.

   b. The types of the base and the exponent may be any combination of real, extended real, and integer.

   c. If the exponent is an integer, the result is as follows:

exp.	base	result
>0	all	same type as base
=0	≠0	same type as base
=0	=0	operation is undefined
<0	≠0	real if base is real, otherwise extended real
<0	=0	operation is undefined

   d. If exponent is real or extended real, the result is as follows:

exp.	base	result
>0, =0, <0	>0	real if base is real, otherwise extended real
>0	=0	real if base is real, otherwise extended real
≤0	=0	operation is undefined
≥0, =0, <0	<0	operation is undefined

5. Hierarchy of Operators

   a. exponentiation ↑

   b. multiplication and division * / %

   c. addition and subtraction + -

6. Expressions inside parentheses are evaluated first.

7. Evaluation proceeds basically from left to right within the hierarchy and within parentheses. Function calls and parenthesized quantities are evaluated from left to right.

'IF' CLAUSE ARITHMETIC EXPRESSION

'IF' a 'THEN' b 'ELSE' c, where a is a Boolean expression, b is a simple arithmetic expression, and c is either a simple arithmetic expression or an 'IF' clause arithmetic expression.

The 'IF' clause arithmetic expression causes one of several arithmetic expressions to be evaluated on the basis of the value of Boolean expressions.

The expression to be evaluated is selected as follows:

1. The Boolean expressions are evaluated one by one in sequence from left to right until one having a value 'TRUE' is found.

2. The value of the 'IF' clause arithmetic expression is the value of the first simple arithmetic expression following this Boolean expression.

ARITHMETIC EXPRESSION

Either a simple arithmetic expression or an 'IF' clause arithmetic expression.

SIMPLE BOOLEAN EXPRESSION

A sequence of variables, subscripted variables, function calls, and relations possibly separated by logical operators and parentheses, which represents a rule for computing a logical value (i.e., 'TRUE' or 'FALSE').

1. Variables and functions used with the logical operators must be declared to be of type Boolean.

2. A relation is composed of two arithmetic expressions separated by a relational operator.

   Example:

   ```
   A-B*C 'EQ' Z*Y
   ```

3. A relation has a value of 'TRUE' if the relation is satisfied; otherwise it has a value of 'FALSE'.

4. The logical operators are defined as follows:

   a. 'NOT' a is true or false if a is false or true, respectively.

   b. a 'AND' b is true if both a and b are true, otherwise it is false.

   c. a 'OR' b is false if both a and b are false, otherwise it is true.

   d. a 'IMP' b is false if a is true and b is false, otherwise it is true.

   e. a 'EQV' b is true if either both a and b are true or both are false, otherwise it is false.

5. The hierarchy of operations in evaluating a Boolean expression is as follows:

   a. arithmetic operators - same order as for arithmetic expressions

   b. relational operators

   c. logical operators

6. The hierarchy of logical operators is:

   a. 'NOT'

   b. 'AND'

   c. 'OR'

   d. 'IMP'

   e. 'EQV'

7. Expressions inside parentheses are evaluated first.

'IF' CLAUSE BOOLEAN EXPRESSION

'IF' a 'THEN' b 'ELSE' c, where b is a simple Boolean expression and a and c are either simple Boolean expressions or 'IF' clause Boolean expressions.

The 'IF' clause Boolean expression is evaluated in the same way as an 'IF' clause arithmetic expression.

BOOLEAN EXPRESSION

Either a simple Boolean expression or an 'IF' clause Boolean expression.

EXPRESSION

Either an arithmetic expression or a Boolean expression.

STATEMENT LABEL

An identifier placed before a statement.

A statement label must be followed by a colon to separate the label from the statement.

A statement may have more than one label, each one followed by a colon.

Statement labels are used so that a statement may be referenced.

Examples:

1. AB: A ← B;
2. AC: 'BEGIN' A ← C 'END'
3. AD: 'BEGIN' AE: A ← E 'END'
4. AF: AG: AH: A ← H;

SWITCH DESIGNATOR

sw[a], where sw represents a switch identifier and a represents an arithmetic expression.

CONDITIONAL DESIGNATOR

A clause of the form 'IF' b 'THEN' c 'ELSE' d, where b represents a Boolean expression, c may be either a statement label, a switch designator, or a conditional designator enclosed within parentheses, and d may be either a statement label, a switch designator, or a conditional designator (which need not be enclosed within parentheses).

DESIGNATIONAL EXPRESSION

A statement label, a switch designator, or a conditional designator.

SIMPLE STATEMENT

A statement which is not a compound statement or a block.

Examples:

assignment
conditional
dummy
'FOR'
'GO TO'
procedure

SECTION IV

STATEMENT AND DECLARATION FORMS

The form of each statement and declaration is listed below. Section V describes each statement in detail; Section VI describes each declaration in detail.

ASSIGNMENT STATEMENTS

Name	Form
Assignment, simple	$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n \leftarrow e$
Assignment, 'IF' clause	$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n \leftarrow$ 'IF' b_1 'THEN' e_1 'ELSE' e_2
Assignment, two 'IF' clauses	$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n \leftarrow$ 'IF' b_1 'THEN' e_1 'ELSE' 'IF' b_2 'THEN' e_2 'ELSE' e_3
Assignment, n 'IF' clauses	$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n \leftarrow$ 'IF' b_1 'THEN' e_1 'ELSE' 'IF' b_2 'THEN' e_2 'ELSE' ... 'IF' b_n 'THEN' e_n 'ELSE' e_{n+1}

CONDITIONAL STATEMENTS

Name	Form
Conditional, simple	'IF' b 'THEN' s
Conditional, 'ELSE'	'IF' b 'THEN' s_1 'ELSE' s_2
Conditional, two 'IF' clauses	'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2
Conditional, two 'IF' clauses, 'ELSE'	'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2 'ELSE' s_3
Conditional, n 'IF' clauses	'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2 'ELSE' ... 'IF' b_{n-1} 'THEN' s_{n-1} 'ELSE' 'IF' b_n 'THEN' s_n
Conditional, n 'IF' clauses, 'ELSE'	'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2 'ELSE' ... 'IF' b_{n-1} 'THEN' s_{n-1} 'ELSE' s_n

DUMMY STATEMENT

Name	Form
Dummy	(null form)

'FOR' STATEMENTS

Name	Form
'FOR', expression	'FOR' v ← e 'DO' s
'FOR', 'STEP' clause	'FOR' v ← e_1 'STEP' e_2 'UNTIL' e_3 'DO' s
'FOR', 'WHILE' clause	'FOR' v ← e 'WHILE' b 'DO' s
'FOR', general	'FOR' v ← $a_1, a_2, \ldots, a_n$ 'DO' s

'GO TO' STATEMENTS

Name	Form
'GO TO', label	'GO TO' a
'GO TO', switch designator	'GO TO' sw [a]
'GO TO', conditional designator	'GO TO' 'IF' b 'THEN' d_1 'ELSE' d_2

PROCEDURE STATEMENT

Name	Form
Procedure statement	name (a_1 t a_2 t ... t a_n)

'ARRAY' DECLARATIONS

Name	Form
'ARRAY'	type 'ARRAY' $a_1, a_2, \ldots, a_n$
'ARRAY', 'OWN'	'OWN' type 'ARRAY' $a_1, a_2, \ldots, a_n$

'PROCEDURE' DECLARATIONS

Name	Form
'PROCEDURE' declaration, simple	'PROCEDURE' name (a_1 t a_2 t ... t a_n); s
'PROCEDURE' declaration, specification part	'PROCEDURE' name (a_1 t a_2 t ... t a_n); sp list; sp list;...; sp list; s
'PROCEDURE' declaration, value and specification part	'PROCEDURE' name (a_1 t a_2 t ... t a_n); 'VALUE' list; sp list; sp list;...; sp list; s
'PROCEDURE' declaration, function definition	type 'PROCEDURE' name (a_1 t a_2 t ... t a_n); 'VALUE' list; sp list sp list;...; sp list; s
'PROCEDURE' declaration, separately compiled	'CODE' 'BEGIN' $d_1 d_2$;...;d_n 'END'

'SWITCH' DECLARATION

Name	Form
'SWITCH'	'SWITCH' sw ← $d_1, d_2, \ldots, d_n$

TYPE DECLARATIONS

Name	Form
Type	type $v_1, v_2, \ldots, v_n$
Type, 'OWN'	'OWN' type $v_1, v_2, \ldots, v_n$

'LINK' DECLARATION

Name	Form
'LINK'	'LINK' string;

SECTION V

STATEMENTS

DESCRIPTIVE FORMAT

The description of each statement in the ALGOL language is presented in this Section. Each statement description starts on a new page, with the format of its descriptive material given as shown below:

Descriptive name

(A brief statement of the purpose of the statement)

Form

(Form of the statement)
(Definition of symbols used in the form line)

Rules

(A list of rules governing the correct usage of the statement; includes restrictions, suggestions, etc.)

Examples

(A list of examples illustrating the use of the statement)

ASSIGNMENT, SIMPLE

To perform numerical calculations; to perform Boolean operations; to assign a value to one or more variables or procedure identifiers in a single statement.

Form

$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n \leftarrow e$

$a_1, a_2, \ldots, a_n$: variables, subscripted variables or procedure identifier

e: arithmetic or Boolean expression

Rules

1. This statement causes expression "e" to be evaluated and the result to be assigned to $a_1, a_2, \ldots, a_n$. (Note. There need be only one variable; e.g., $a_1 \leftarrow e$).

2. The character "$\leftarrow$" signifies assignment of the value of the expression to the variable(s).

3. The process of assignment is as follows:

   a. Subscripts, if any, occurring in the variables are evaluated from left to right.

   b. The expression "e" is evaluated.

   c. The value of the expression is assigned to all the variables $a_1, a_2, \ldots, a_n$ from right to left across the left side as follows: The value of e is assigned to a_n, the value of a_n is assigned to a_{n-1}, etc. Finally, the value of a_2 is assigned to a_1.

4. The types of the variables must be as follows:

   a. The types may be all Boolean. In this case, the expression "e" must be Boolean.

   b. The types may be real, extended real, or integer. In this case, the expression "e" must be arithmetic.

   c. Boolean types may not be mixed with the other types.

5. When "e" is an arithmetic expression and its type and the type of variable a_n is different, the value of "e" is changed to the type specified by a_n before it is assigned to a_n. (See "ALGOL Language Definitions", Section III, for forms of integers, real numbers, and extended real numbers.)

6. In the case in which "e" is real or extended real and a_n is an integer, "e" is operated upon by the function ENTIER (e+.5). The result of ENTIER is the largest integer not greater than the value of the argument. This value is then assigned to a_n.

ASSIGNMENT

7. When the type of a_i and a_{i-1} is different, the value of a_i, is changed before it is assigned to a_{i-1}.

8. The case of an a_i being a procedure identifier is only used in defining functions. (See 'PROCEDURE' declaration, function definition.)

Examples

In these examples, A, B, C, and D identify 'REAL' type variables. R AND S identify 'INTEGER' type variables, and W identifies a 'BOOLEAN' type variable.

1. A ← B+C

   The value of B + C is assigned to A.

2. A ← D ← B+C

   The value of B + C is assigned to A and D.

3. A ← R ← 3.9

   4 is assigned to R and A.

4. R ← A ← 3.9

   3.9 is assigned to A and 4 is assigned to R.

5. J ← 1; S[J] ← J ← 2

   First, 1 is assigned to J. Then 2 is assigned to J and S[1].

6. W ← A 'GR' B

   If the value of A is greater than the value of B, W is assigned the value 'TRUE'; otherwise, W is assigned the value 'FALSE'.

ASSIGNMENT, 'IF' CLAUSE

To permit a choice to be made as to which of two expressions is to be evaluated, based on the value of a Boolean expression; to assign the value of the evaluated expression to one or more variables or procedure identifiers.

Form

$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n$ 'IF' b 'THEN' e_1 'ELSE' e_2

$a_1, a_2, \ldots, a_n$: variables, subscripted variables or procedure identifier

b: Boolean expression

e_1, e_2: arithmetic or Boolean expression

Rules

1. Subscripts, if any, occurring in the variables $a_1, a_2, \ldots, a_n$ evaluated from left to right.
2. The Boolean expression "b" is evaluated.
3. If the value of b is 'TRUE' expression, e_1 is evaluated; if 'FALSE', e_2 is evaluated.
4. After e_1 or e_2 is evaluated, this statement operates as a simple assignment statement with the evaluated expression.

Examples

1. P ← 'IF' Q 'LS' 10.0 'THEN' R 'ELSE' S + 17.5

   If Q < 10, P receives the value of R, otherwise S + 17.5.

2. A ← B ← C ← 'IF' D 'THEN' E 'OR' F 'ELSE' G 'AND' H

   If D is true, the value of E 'OR' F is assigned to A, B, and C. Otherwise, the value of G 'AND' H is assigned.

ASSIGNMENT, TWO 'IF' CLAUSES

To permit a choice to be made as to which of three expressions is to be evaluated, based on the values of two Boolean expressions; to assign the value of the evaluated expression to one or more variables or procedure identifiers.

Form

$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n \leftarrow$ 'IF' b_1 'THEN' e_1 'ELSE' 'IF' b_2 'THEN' e_2 'ELSE' e_3

$a_1, a_2, \ldots, a_n$: variables, subscripted variables or procedure identifier

b_1, b_2: Boolean expressions

e_1, e_2, e_3: arithmetic or Boolean expressions

Rules

1. Subscripts, if any, occurring in the variables $a_1, a_2, \ldots, a_n$ are evaluated from left to right.
2. The Boolean expression "b_1" is evaluated.
3. If b_1 is true, e_1 is evaluated; if b_1 is false, b_2 is evaluated.
4. If b_2 is true, e_2 is evaluated; if b_2 is false, e_3 is evaluated.
5. After an expression is evaluated, this statement operates as a simple assignment statement with the evaluated expression.

Example

```
R ← 'IF' T 'THEN' B-6.2 'ELSE' 'IF' U 'THEN' C-7 'ELSE' D%3.5
```

If T is true, R is assigned the value of B-6.2. If T is false and U is true, C-7 is assigned to R. Otherwise, D%3.5 is assigned to R.

ASSIGNMENT, n 'IF' CLAUSES

To permit a choice to be made as to which of a number of expressions is to be evaluated, based on the value of Boolean expressions; to assign the value of the evaluated expression to one or more variables or procedure identifiers.

Form

$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_n$ 'IF' b_1 'THEN' e_1 'ELSE' 'IF' b_2 'THEN' e_2 'ELSE' ... 'IF' b_n 'THEN' e_n 'ELSE' e_{n+1}

$a_1, a_2, \ldots, a_n$: variables, subscripted variables or procedure identifier

$b_1, b_2, \ldots, b_n$: Boolean expressions

$e_1, e_2, \ldots, e_{n+1}$: arithmetic or Boolean expressions

Rules

1. Subscripts, if any, occurring in the variables $a_1, a_2, \ldots, a_n$ are evaluated from left to right.
2. The Boolean expressions $b_1, b_2, \ldots, b_n$ are evaluated from left to right until one is found which has a value of 'TRUE'.
3. If b_i is found to be true, then e_i is evaluated.
4. If all the Boolean expressions are false, e_{n+1} will be evaluated.
5. After step 3 or 4 above, this statement operates as a simple assignment statement with the evaluated expression.

Example

```
C [D 4,2,2] ← 'IF' B 'OR' E 'THEN' 5 'ELSE' 'IF' T 'THEN'
7.5 'ELSE' 'IF' A 'LS' C 'THEN' G 'ELSE' L
```

C and D [4,2,2] may be assigned the following values: 5 if either B or E is true; 7.5 if T is true; the value of G if the value of A is less than the value of C; the value of L if none of the above conditions are true.

CONDITIONAL, SIMPLE

To permit a statement to be executed or skipped, depending on the value of a Boolean expression.

Form

'IF' b 'THEN' s

b: Boolean expression

s: statement

Rules

1. Statement s may be any one of the following:
   a. assignment statement
   b. 'GO TO' statement
   c. dummy statement
   d. 'FOR' statement
   e. procedure statement
   f. compound statement
   g. block
2. Statement s may have a label.
3. If the Boolean expression has a value of 'TRUE', statement s is executed. If s does not explicitly specify its successor, the statement following will be executed next.
4. If the Boolean expression has a value of 'FALSE', statement s is skipped and the following statement will be executed next.

CONDITIONAL

Examples

1. 'IF' A 'GR' B 'THEN' D ← E*F

   If the value of A is greater than the value of B, then the value of E*F is assigned to D. Otherwise, the assignment statement is skipped and the statement following it is executed.

2. 'IF' L 'THEN' 'BEGIN' P ← P+3; R ← 17.5-T; L ← 'FALSE' 'END'; 'GO TO' S9

   If L is true, the compound statement enclosed between 'BEGIN' and 'END' will be executed; followed by 'GO TO' S9; if L is false, only 'GO TO' S9 will be executed.

CONDITIONAL, 'ELSE'

To permit a choice to be made as to which one of two specified statements is to be executed. The decision is based on the value of a Boolean expression.

Form

'IF' b 'THEN' s_1 'ELSE' s_2

b: Boolean expression

s_1, s_2: statements

Rules

1. The statements s_1 and s_2 may be any one of the following:
   a. assignment statement
   b. 'GO TO' statement
   c. procedure statement
   d. dummy statement
   e. compound statement
   f. block
2. Statement s_2 may also be a 'FOR' statement.
3. Statements s_1 and s_2 may be labelled.
4. If the Boolean expression has a value of 'TRUE', statement s_1 is executed. If s_1 does not explicitly specify its successor, then the statement following the conditional statement is executed next; i.e., s_2 is skipped.
5. If the Boolean expression has a value of 'FALSE', statement s_2 is executed. If s_2 does not explicitly specify its successor the statement following the conditional statement is executed next.

CONDITIONAL

Examples

1. 'IF' A 'LS' B 'THEN' T ← T+1 'ELSE' B ← B+1; 'GO TO' L1

   If A is less than B, T ← T+1 is executed, followed by 'GO TO' L1. If A is greater than or equal to B, B ← B+1 is executed, followed by 'GO TO' L1.

2. 'IF' R 'AND' S 'THEN' 'GO TO' BOB 'ELSE' JOE: M ← N+P; 'GO TO' BOB

   If the expression is true, control is transferred to the statement labelled BOB; if false, the statement labelled JOE is executed and then control goes to the statement labelled BOB.

CONDITIONAL

CONDITIONAL, TWO 'IF' CLAUSES

To permit a choice to be made as to which of two statements is to be executed or whether neither is to be executed, depending on the values of two Boolean expressions.

Form

'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2

b_1,b_2: Boolean expressions

s_1,s_2: statements

Rules

1. Statements s_1 and s_2 may be any one of the following:
   a. assignment statement
   b. 'GO TO' statement
   c. dummy statement
   d. procedure statement
   e. compound statement
   f. block
2. Statement s_2 may also be a 'FOR' statement.
3. Statements s_1 and s_2 may be labelled.
4. If b_1 has a value of 'TRUE', statement s_1 is executed. If s_1 does not explicitly specify its successor, the statement following the conditional statement is executed next.
5. If b_1 has a value of 'FALSE', b_2 is evaluated.
6. If b_2 has a value of 'TRUE', statement s_2 is executed. If s_2 does not explicitly specify its successor, the statement following the conditional statement is executed next.
7. If b_2 has a value of 'FALSE', then s_2 is skipped and the statement following the complete conditional statement is executed next.

CONDITIONAL

Example

```
'IF' A 'EQ' B 'THEN' MODE (C,D) 'ELSE' 'IF' A 'GR' B
'THEN' MEAN (T,D); R←D*F
```

If A=B, procedure MODE is executed, followed by R←D*F. If A≠B but A>B, then procedure MEAN is executed followed by R←D*F. If A<B, then only R←D*F is executed.

CONDITIONAL, TWO 'IF' CLAUSES, 'ELSE'

To permit a choice to be made as to which of three statements is to be executed, depending upon the value of two Boolean expressions.

Form

'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2 'ELSE' s_3

b_1, b_2: Boolean expressions

s_1, s_2, s_3: statements

Rules

1. Statements s_1, s_2, and s_3 may be any one of the following:
   a. assignment statement
   b. 'GO TO' statement
   c. dummy statement
   d. procedure statement
   e. compound statement
   f. block
2. Statement s_3 may also be a 'FOR' statement.
3. Statements s_1, s_2, and s_3 may be labelled.
4. If b_1 has a value of 'TRUE', statement s_1 is executed. If s_1 does not explicitly specify its successor, the statement following the conditional statement is executed next.
5. If b_1 is false, b_2 is evaluated.
6. If b_2 has a value of 'TRUE', statement s_2 is executed. If s_2 does not explicitly specify its successor, the statement following the conditional statement is executed next.
7. If b_2 has a value of 'FALSE', statement s_3 is executed. If s_3 does not explicitly specify its successor, the statement following the conditional statement is executed next.

CONDITIONAL

Example

```
'IF' L 'THEN' 'GO TO' BOY 'ELSE' 'IF' R 'GR' S 'THEN'
'BEGIN' A ← A+1; CALC (F,10) 'END' 'ELSE' 'GO TO' CAT; R ← R+1
```

If L is true, control goes to the statement labelled BOY; if L is false, R is compared to S; if R>S, the compound statement is executed followed by R ← R+1. If R≤S, control goes to the statement labelled CAT.

CONDITIONAL

CONDITIONAL, n 'IF' CLAUSES

To permit a choice to be made among a number of statements as which one should be executed, or whether none is to be executed, depending upon the value of Boolean expressions.

Form

'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2 'ELSE' ...
'IF' b_{n-1} 'THEN' s_{n-1} 'ELSE' 'IF' b_n 'THEN' s_n

$b_1, b_2, \ldots, b_n$: Boolean expressions

$s_1, s_2, \ldots, s_n$: statements

Rules

1. Each statement $s_1, s_2, \ldots, s_n$ may be any one of the following:
   a. assignment statement
   b. 'GO TO' statement
   c. dummy statement
   d. procedure statement
   e. compound statement
   f. block
2. Statement s_n may be a 'FOR' statement.
3. Statements $s_1, s_2, \ldots, s_n$ may be labelled.
4. The Boolean expressions are evaluated in the order $b_1, b_2, \ldots,$ until one having a value of 'TRUE' is found. If b_i is true, statement s_i is executed. If statement s_i does not explicitly specify its successor, the statement following the conditional statement is executed next.
5. If none of the Boolean expressions is true, the statement following the complete conditional statement is executed next.

CONDITIONAL

Example

```
'IF' M 'THEN' A←A+1 'ELSE' 'IF' N 'THEN' 'GO TO' R1
'ELSE' 'IF' P 'THEN' 'FOR' 1  1 'STEP' 1 'UNTIL' 10 'DO'
A[I] ←  I; L←M 'OR' P
```

If M is true, the value of A is increased by 1. If M is false and N is true, then 'GO TO' R1 is executed. If M and N are false and P is true, the 'FOR' statement is executed. If M, N and P are all false, the statement L←M 'OR' P is executed.

CONDITIONAL

CONDITIONAL, n 'IF' CLAUSES, 'ELSE'

To permit a choice to be made among a number of statements as to which one should be executed, depending upon the value of Boolean expressions.

Form

'IF' b_1 'THEN' s_1 'ELSE' 'IF' b_2 'THEN' s_2 'ELSE' ...
'IF' b_{n-1} 'THEN' s_{n-1} 'ELSE' s_n

$b_1, b_2, \ldots, b_n$: Boolean expressions

$s_1, s_2, \ldots, s_n$: statements

Rules

1. Each statement $s_1, s_2, \ldots, s_n$ may be any one of the following:
   a. assignment statement
   b. 'GO TO' statement
   c. dummy statement
   d. procedure statement
   e. compound statement
   f. block
2. Statement s_n may be a 'FOR' statement.
3. Statements $s_1, s_2, \ldots, s_n$ may be labelled.
4. The Boolean expressions are evaluated in the order $b_1, b_2, \ldots,$ until one having a value of 'TRUE' is found. If b_i is true, statement s_i is executed. If statement s_i does not explicitly specify its successor, the statement following the complete conditional statement is executed next.
5. If none of the Boolean expressions is true, statement s_n will be executed. If it does not explicitly specify its successor, the statement following the conditional statement is executed next.

CONDITIONAL

Example

```
'IF' A "THEN' I←I+1 'ELSE' 'IF' B 'THEN' J←J+1 'ELSE'
'IF' C 'THEN' K←K+1 'ELSE' L←L+1; 'IF' D 'THEN'
'GO TO' BAD
```

If A is true, the value of I is increased by one. If A is false and B is true, the value of J is increased by one. If A and B are false and C is true, the value of K is increased by one. If A, B, and C are false, the value of L is increased by one. If D is true then 'GO TO' BAD is executed. Otherwise, the statement following it is executed.

DUMMY

DUMMY STATEMENT

To place a label at a particular point in the program.

Form

(null form)

Rule

This statement causes no operation.

Examples

1. COUNT:;

   COUNT is the label of a dummy statement.

2. B3: ; ABC: E ← E+1

   B3 is the label of a dummy statement.

3. 'BEGIN'...; TOY: 'END'

   TOY is the label of a dummy statement.

'FOR'

'FOR', EXPRESSION

To permit a statement to be executed for a specified value of a controlled variable.

Form

'FOR' v← e 'DO' s

v: variable or subscripted variable

e: arithmetic expression

s: statement

Rules

1. Variable v is called the controlled variable of the 'FOR' statement.
2. e represents a value which is assigned to v.
3. Statement s may be a simple statement, a compound statement, or a block.
4. The 'FOR' statement causes the expression e to be evaluated and its value assigned to v. Then statement s is executed.
5. After statement s is executed with v having the value of e, the 'FOR' statement has been executed. If s does not explicitly specify its successor, the statement following the 'FOR' statement is executed next.
6. After execution of the 'FOR' statement, the value of v is undefined.
7. If control is transferred from the 'FOR' statement by a statement (within statement s), the value of v is available.
8. A 'GO TO' statement outside the 'FOR' statement may not refer to a label within the 'FOR' statement.

Examples

1. 'FOR' J←1 'DO' A[J]←0.0

   This statement causes zero to be assigned to location A[I].

2. 'FOR' R 2*BOY↑2 'DO' 'BEGIN' T←T+1;
   B[R]←-C[R] 'END'

   This statement results in -C[2* BOY↑2] assigned to B[2*BOY↑2]. Also, the value of T is increased by one.

'FOR', 'STEP' CLAUSE

To permit a statement to be executed repeatedly for a specified initial value, increment, and final value of a controlled variable.

Form

'FOR' v ← e_1 'STEP' e_2 'UNTIL' e_3 'DO' s

v: variable or subscripted variable

e_1,e_2,e_3: arithmetic expression(s)

s: statement

Rules

1. Variable v is called the controlled variable of the 'FOR' statement.

2. e_1 represents the initial value for v; e_2 is the increment of v; e_3 is the final value for v.

3. Statement s may be a simple statement, a compound statement or a block.

4. The first step in the operation of the 'FOR' statement is that v is assigned the value of e_1.

5. Statement s may be executed a number of times as follows:

   a. A test is made to see if the value of v is beyond the bound specified by e_3. If it is, statement s will not be executed. The statement after s is executed next and the value of v is undefined.

   b. If v is within the bound, statement s is executed.

   c. If s does not explicitly specify its successor, the value e_2 is then added to v (i.e., v ← v+e_2). If the value of e_2 is positive, this will have the effect of increasing v. If the value of e_2 is negative, v will be reduced. The process is then repeated at step a.

6. If control is transferred from the 'FOR' statement by a statement (within statement s), the value of v is available.

7. The value of the controlled variable, the increment and the final value may be changed by statement s. Therefore, they are evaluated every time reference is made to them.

8. A 'GO TO' statement outside a 'FOR' statement may not refer to a label within the 'FOR' statement.

Examples

1. `'FOR' I←1 'STEP' 1 'UNTIL' 10 'DO' A[I]←B[I]`

   These statements cause B[1] to B[10] to be assigned to A[1] to A[10].

2. `'FOR' K←9 'STEP' -2 'UNTIL' 5 'DO' X[K]←K↑2`

   These statements cause 81 to be assigned to X[9], 49 to be assigned to X[7], and 25 to be assigned to X[5].

3. ```
   'FOR' L←1 'STEP' 1 'UNTIL' 5 'DO' 'BEGIN'
   'FOR' A[L]←6 'STEP' 1 'UNTIL' 10 'DO' B[A[L],L]←L
   'END'
   ```

   The order of assignments caused by these statements is as follows:

   6 to 10 is assigned to A[1] as 1 is assigned to B[6,1] to B[10,1],
   6 to 10 is assigned to A[2] as 2 is assigned to B[6,2] to B[10,2],
   etc.
   Finally, 6 to 10 is assigned to A[5] as 5 is assigned to B[6,5] to B[10,5].
   ```

'FOR'

'FOR', 'WHILE' CLAUSE

To permit a statement to be executed repeatedly for assigned values of a controlled variable, with repetition controlled by the value of a Boolean expression.

Form

'FOR' v ← e 'WHILE' b 'DO' s

v: variable or subscripted variable

e: arithmetic expression

b: Boolean expression

s: statement

Rules

1. Variable v is called the controlled variable of the 'FOR' statement.
2. Statement s may be a simple statement, a compound statement or a block.
3. This statement causes statement s to be executed repeatedly as long as the value of the Boolean expression b is true.
4. This statement operates as follows:
   a. e is evaluated and its value is assigned to v.
   b. The Boolean expression b is evaluated.
   c. If b is true, statement s is executed. If s does not explicitly specify its successor, the process is repeated at step a.
   d. If b is false, statement s is not executed and the statement following statement s is executed next. The value of v is undefined in this case.
5. If control is transferred from the 'FOR' statement by a 'GO TO' statement (within statement s), the value of v is available.
6. The values of either e or b may be changed by statement s.
7. A 'GO TO' statement outside a 'FOR' statement may not refer to a label within the 'FOR' statement.

Example

```
J←1; 'FOR' I←J 'WHILE' I 'LS' 10 'DO' 'BEGIN'
A[I]←I; J←J+1 'END'
```

These statements cause 1 to 9 to be assigned to A[1] to A[9].

'FOR'

'FOR' GENERAL

To permit a statement to be executed repeatedly for various conditions governing a controlled variable.

Form

'FOR' v ← a_1,a_2,...,a_n 'DO' s

v: variable or subscripted variable

a_1,a_2,...,a_n: arithmetic expression, 'STEP' clause, or 'WHILE' clause

s: statement

Rules

1. Variable v is called the controlled variable of the 'FOR' statement.
2. a_1,a_2,...,a_n may be any combination of arithmetic expressions, 'STEP' clauses, or 'WHILE' clauses.
3. s may be a simple statement, a compound statement or a block.
4. If a_i is an arithmetic expression, a 'STEP' clause or a 'WHILE' clause, the 'FOR' statement operates as previously described. The order of operation is a_1,a_2,...,a_n.

Example

'FOR' X ← 3, 2 'STEP' 1 'UNTIL' 5, 70, 60, A 'WHILE' Z, 80 'DO' P(X)

First, 3 is assigned to X and procedure P(X) is executed.

Then the 'STEP' clause causes the following action: 2 is assigned to X and P(X) is executed. X is stepped by 1 three times causing it to assume the values 3, 4, and 5. P(X) is executed after each step of X. Next, X is assigned the value 70, and P(X) is executed.

Then X is assigned the value 60, and P(X) is executed.

The 'WHILE' clause causes the value of A to be assigned to X. If Z is true, P(X) is executed. This is repeated until Z becomes false. (The values of A and Z may be changed by execution of P(X)).

Finally, 80 is assigned to X and P(X) is executed.

'GO TO'

'GO TO', LABEL

To interrupt the normal sequence of statement execution by defining explicitly the successor of the current statement.

Form

'GO TO' a

a: statement label

Rules

1. The statement, 'GO TO' a, causes control to go to the statement with label a.
2. A 'GO TO' statement outside a 'FOR' statement may not refer to a label within the 'FOR' statement.
3. A 'GO TO' statement outside a block may not refer to a label within that block.
4. A 'GO TO' statement outside a compound statement may refer to label within that compound statement.

Examples

1. 'GO TO' BOY

   This statement causes control to go to a statement labelled BOY.

2. 'GO TO' T12; M15: A←A+1; 'IF' L 'THEN' 'BEGIN'
   C D*E↑2; T12: A←B+C*F 'END'

   The 'GO TO' statement causes control to go to a statement within a compound statement.

'GO TO'

'GO TO', SWITCH DESIGNATOR

To interrupt the normal sequence of statement execution by causing control to be transferred to one of a number of possible statements, depending on the value of an arithmetic expression.

Form

'GO TO' sw [a]

sw: switch identifier

a: arithmetic expression

Rules

1. The switch identifier "sw" must have been defined by a switch declaration in the current block or in an enclosing block.

2. The form sw [a] is called a switch designator.

3. The next statement to be executed is the one whose label is referenced through the switch declaration defining "sw".

4. This 'GO TO' statement operates as follows:

   a. The expression denoted by a is evaluated. From this value an integer k is established where k is the result of the function ENTIER (a+.5). That is, the largest integer not greater than the value of the argument, i.e., if a is 3.7, k=4.

   b. k specifies which element in the list of the switch declaration will be referenced; i.e., the leftmost element is numbered 1; the next is 2, etc.

   c. If k is not within the range 1 to n (where n is the number of elements in the switch designator), control goes to the next statement in normal sequence.

5. A 'GO TO' statement outside a 'FOR' statement may not refer to a label within that 'FOR' statement.

6. A 'GO TO' statement outside a block may not refer to a label within that block.

7. A 'GO TO' statement outside a compound statement may refer to a label within that compound statement.

Example

```
'BEGIN' 'SWITCH' AB ← PB, QB;
'SWITCH' AC ← PC, QC, AB[X];
 ...
'GO TO' AB[T];
 ...
'GO TO' AC[Y];
 ...
'END'
```

If T has the value 1 when the 'GO TO' for switch AB is executed, control goes to the statement labelled PB. If T has the value 2, control goes to the statement labelled QB. If T has any other value, control goes to the statement following the 'GO TO' statement. When the 'GO TO' for switch AC is executed, control will go to statements labelled PC or QC if Y has the value one or two, respectively. If Y has the value three, then execution is equivalent to 'GO TO' AB[X]. If Y has any other value, control goes to the next sequential statement.

'GO TO', CONDITIONAL DESIGNATOR

To interrupt the normal sequence of statement execution by causing control to be transferred to one of a number of possible statements; the statement chosen will depend on the value of a Boolean expression.

Form

'GO TO' 'IF' b 'THEN' d_1 'ELSE' d_2

b: Boolean expression

d_1,d_2: designational expressions

Rules

1. A designation expression (d_1,d_2) is any one of the following:
   a. Statement label
   b. Switch designator. This has the form sw[a], where sw represents a switch identifier and a represents an arithmetic expression.
   c. Conditional designator. This has the form

      'IF' b 'THEN' c 'ELSE' d

      where

      b represents a Boolean expressions;

      c may be either a statement label, a switch designator, or a conditional designator enclosed within parentheses;

      d may be either a statement label, a switch designator, or a conditional designator (not necessarily enclosed within parentheses).
2. This statement operates as follows:
   a. The Boolean expression b is evaluated;
   b. If b is true, control is transferred as specified by d_1.
   c. If the Boolean expression b is false, control is transferred as specified by d_2.
3. A 'GO TO' statement outside a 'FOR' statement may not refer to a label within that 'FOR' statement.
4. A 'GO TO' statement outside a compound statement may refer to a label within that compound statement.

'GO TO'

Examples

1. 'GO TO' 'IF' A 'THEN' B 'ELSE' C[I]

   If the Boolean expression A is true, control goes to the statement labelled B. Otherwise, control goes to the statement referenced by the Ith item in the switch declaration defining C.

2. 'GO TO' 'IF' BA 'THEN' LA 'ELSE' 'IF' BB 'THEN' LB 'ELSE' LC

   If the Boolean expression BA is true, control goes to the statement labelled LA. If expression BA is false and Boolean expression BB is true, control goes to the statement labelled LB. If both expressions BA and BB are false, control goes to the statement labelled LC. (Note: d_1 in this case is a statement label while d_2 is a conditional designator.)

3. 'GO TO' 'IF' BA 'THEN' ('IF' BB 'THEN' LB 'ELSE' LC) 'ELSE' LA

   If both BA and BB are true, control goes to the statement labelled LB. If BA is true and BB is false, control goes to the statement labelled LC. If BA is false, control goes to the statement labelled LA. (Note. d_1 is a conditional designator, and therefore, must be enclosed in parentheses.)

PROCEDURE STATEMENT

To call for the execution of a procedure defined by a 'PROCEDURE' declaration.

Form

(1) name

(2) name (a_1 t a_2 t ... t a_n)

name: procedure identifier

$a_1, a_2, \ldots, a_n$: actual parameters

t: separator

Rules

1. A procedure statement may have no parameters, as shown in Form (1).

2. When there are parameters (Form (2)), each separator t may be either "," or ")b:(" where b is only descriptive; i.e., it may be used as comments to describe actual parameters. b has no operational significance.

3. The procedure identifier must appear in a procedure declaration.

4. The number of actual parameters must be the same as the number of formal parameters in the procedure declaration. However, the method of parameter separation need not be the same in a procedure statement and the corresponding declaration. That is, where a comma was used in a procedure statement, the form ")b:(" may be used in the declaration and vice versa.

5. The actual parameters may be any one of the following:

   a. arithmetic expression

   b. Boolean expression

   c. string

   d. array identifier

   e. switch identifier

   f. procedure identifier

   g. designational expression

6. The correspondence between the actual parameters of the procedure statement and the formal parameters of the procedure declaration is by their appearance in the respective parameter lists. The two sets of parameters must have the same number of items.

7. The execution of a procedure statement is as follows:

   a. The formal parameters which appear in a value list of the procedure declaration are replaced by the values of the corresponding actual parameters.

   b. These actual parameters are evaluated from left to right according to their appearance in the parameter list.

   c. Formal parameters which are not part of a value list are replaced throughout the procedure by the corresponding actual parameters.

   d. If the identifier of an actual parameter and an identifier already in the procedure are the same, adjustments will automatically be made to the latter so that no conflicts occur.

   e. After the procedure has been modified as above, it is executed.

8. If an actual parameter is a string, it may only be used in a procedure written in non-ALGOL code. In an ALGOL procedure, a string may appear only as an actual parameter for a further procedure call.

10. If a formal parameter is an array identifier, the corresponding actual parameter must also be an array identifier of the same dimension.

11. A switch identifier or string may not be an actual parameter corresponding to a formal parameter which is called by value. A procedure identifier may not be used as a value parameter unless it designates a function with no arguments.

Examples

1. HIGHVAL (Z, P*(P+1)/2, V, I)

   The procedure which this statement calls is defined in the section 'PROCEDURE' declaration, simple. In this procedure statement Z denotes the number of elements. The value of the largest element of Z will be found in V after the procedure call, and I will contain the value of the subscript of the largest element.

2. SQUAREROOT (A↑2+B↑2, .000001, C)

   The procedure which this statement calls is defined in the section 'PROCEDURE' declaration, specification part. After this procedure statement is executed, C will contain the square root of A +B with an accuracy of .000001.

PROCEDURE

3. TOT (X, A, 1, N, 1/A*(A+1))

The procedure which this statement calls is defined in the section 'PROCEDURE' declaration, value and specification part. This procedure statement will result in the following computation:

$$X = \sum_{A=1}^{N} 1/A(A+1)$$

4. SUM ← ADD (A, I, N) FUNCTION: (1/A*(A+1))

The procedure which this statement calls is defined in the section 'PROCEDURE' declaration, function definition. This function call will result in the summation of example 3 in ADD and in SUM. The symbol FUNCTION is used as text and has no operational significance.

5. SUM ADD(P,Q,N*(N+1), ADD(Q,1,N,P/Q))

This statement results in the value of the following computation placed in ADD and in SUM:

$$\sum_{P=Q}^{N(N+1)} \sum_{Q=1}^{N} P/Q$$

This is an example of a recursive procedure call.

SECTION VI

DECLARATIONS

DESCRIPTIVE FORMAT

The description of each declaration in the ALGOL language is presented in this Section. Each declaration description starts on a new page, with the format of its descriptive material given as shown below:

Descriptive name

(A brief statement of the purpose of the declaration)

Form

(Form of the declaration)
(Definition of symbols used in the form line)

Rules

(A list of rules governing the correct usage of the declaration; includes restrictions, suggestions, etc.)

Examples

(A list of examples illustrating the use of the declaration)

'ARRAY'

To specify array identifiers, dimensions, bounds of subscripts and array types.

Form

type 'ARRAY' $a_1,a_2,\ldots,a_n$

type: type word

$a_1,a_2,\ldots,a_n$: array specifier(s)

Rules

1. The type word may be any one of the following:
   a. 'INTEGER'
   b. 'REAL'
   c. 'EXTENDED REAL'
   d. 'BOOLEAN'
2. Type is optional. If it is not used, 'REAL' is assumed. The type is assigned to each array identifier in the declaration.
3. An array specifier may be either of the form b or b[c], where b represents an array identifier and c represents a dimension specifier. A dimension specifier has the form d_1: e_1,d_2: e_2,...,d_n: e_n, where each d_i and e_i may be an arithmetic expression. n is the number of dimensions. d_i and e_i represent the lower and upper subscript bounds of dimension i, respectively. The value of a lower bound may not exceed the value of an upper bound.
4. If an array identifier does not have a dimension specifier, the next dimension specifier is assigned. That is, the form $b_1,b_2,\ldots,b_n$ [d_1: e_1, d_2: e_2,...,d_n: e_n] is equivalent to the form b_1 [d_1: e_1, d_1: e_2,...,d_n: e_n], b_2 [d_1: e_1, d_2: e_2,...,d_n: e_n],... b_n [d_1: e_1, d_2: e_2,...,d_n: e_n].
5. Lower and upper bounds will be evaluated from left to right. The bounds can only depend on variables and procedures which have been defined in a block enclosing the block for which the array declaration is valid. Consequently, in the outermost block of a program, only array declarations with constant bounds may be used.
6. The bounds will be evaluated each time the block is entered.
7. Every array used in a program must appear in an array declaration.
8. An array identifier may not appear with subscripts whose values do not lie within the bounds specified by the array declaration.

'ARRAY'

Examples

1. 'ARRAY' A[1:10]

   The array A is one-dimensional and has a lower subscript bound of 1 and an upper subscript bound of 10. A is assumed to be of 'REAL' type.

2. 'ARRAY' A,B [1:10,1:20]

   Arrays A and B are two-dimensional and have subscript bounds 1 and 10 and 1 and 20. The arrays are assumed to be 'REAL' type.

3. 'INTEGER' 'ARRAY' A[P:Q], B [1:2*P, 3:5, 1:5]

   The array A is of 'INTEGER' type and has subscript bounds P and Q. B is of 'INTEGER' type and is three-dimensional. The bounds of the dimensions are 1 and 2*P, 3 and 5, and 1 and 5, respectively.

'ARRAY', 'OWN'

To specify array identifiers, dimensions, bounds of subscripts and array types; also to specify the condition of arrays upon re-entry into a block.

Form

'OWN' type 'ARRAY' $a_1,a_2,\ldots,a_n$

type: type word

$a_1,a_2,\ldots,a_n$: array specifier(s)

Rules

1. The array specifiers may be in any of the forms permissible for the array declaration.
2. All the rules which pertain to array declarations are valid for the 'OWN' array declaration except:
   a. On re-entry into the block in which the 'OWN' array declaration appears the array elements will have their previous values.
   b. The subscript bounds must be integer constants.
3. When exit is made from the block (by 'END' or by a 'GO TO' statement), the identifiers are inaccessible even though their values have been saved.

Example

```
'OWN' 'BOOLEAN' 'ARRAY' BA[1:20, 5:15, 1:10]
```

The array BA is three-dimensional and is of 'BOOLEAN' type. The bounds of the dimensions are 1 and 20, 5 and 15, and 1 and 10, respectively.

'PROCEDURE' DECLARATION, SIMPLE

To define a statement or series of statements as being associated with a procedure identifier; to provide a means by which a procedure may be executed any number of times in the course of a program although the steps of the procedure appear only once.

Form

1) 'PROCEDURE' name; s

2) 'PROCEDURE' name (a_1 t a_2 t ... t a_n); s

name: procedure identifier

$a_1, a_2, \ldots, a_n$: formal parameters

t: separator

s: statement

Rules

1. A procedure declaration may have no parameters, as shown in Form (1).

2. When there are parameters (Form (2)), each separator t may be either "," or ")b:(" where b represents any sequence of letters. The function of b is only descriptive, i.e., it may be used as comments to describe actual parameters. b has no operational significance.

3. The formal parameters may be any of the following:

   a. variable

   b. array identifier

   c. switch identifier

   d. label

   e. procedure identifier

4. The formal parameters usually appear somewhere in statement s. They will be replaced by or assigned the values of the actual parameters of the particular procedure statement which calls the procedure.

5. Statement s may be

   a. a simple statement

   b. a compound statement

   c. a block

'PROCEDURE'

6. Identifiers which are not formal parameters may appear in s if either of the following conditions exists:

   a. s is in the form of a block and the identifiers are declared at the beginning of this block.

   b. the identifiers are declared in the block in which the procedure declaration appears.

7. Statement s always acts like a block insofar as the scope of its identifiers is concerned; i.e., a label appearing in s is not defined outside the procedure declaration.

8. The procedure specified may be executed anywhere in the block in which the declaration appears by writing a procedure statement containing the procedure identifier and the actual parameters, if any.

Example

```
'PROCEDURE' HIGHVAL (A,N) ANS:(X,Y);
   'BEGIN'
      X←A[1]; Y←1; 'FOR' I←2 'STEP' 1 'UNTIL' N 'DO'
   'IF' A[I] 'GR' X 'THEN'
      'BEGIN'
         X←A[I]; Y←I
      'END'
   'END'
```

This procedure determines the largest element of an array. Input formal parameters are: array identifier A and number N of elements. Output formal parameters are: value X of largest element and value Y of subscript of largest element. The symbol ANS is used as text and has no operational significance.

'PROCEDURE' DECLARATION, SPECIFICATION PART

To define a statement or series of statements as being associated with a procedure identifier; to provide a means by which a procedure may be executed any number of times in the course of a program although the steps of the procedure appear only once; to specify the kinds of quantities actual parameters may represent.

Form

'PROCEDURE' name (a_1 t a_2 t ... t a_n)

sp list; sp list;...; sp list; s

name: procedure identifier

$a_1, a_2, \ldots, a_n$: formal parameters

t: separator

sp: specifier

list: formal parameters separated by commas

s: statement

Rules

1. Each separator t may be either "," or ")b:(" where b represents any sequence of letters. The function of b is only descriptive; i.e., it may be used as comments to describe actual parameters. b has no operational significance.

2. The formal parameters may be any of the following:

   a. variable

   b. array identifier

   c. label

   d. switch identifier

   e. procedure identifier

3. The formal parameters usually appear somewhere in statement s. They are replaced at the time of execution by the actual parameters of the procedure statement.

'PROCEDURE'

4. The specifiers may be any of the following:

'ARRAY'	'INTEGER' 'ARRAY'
'BOOLEAN'	'INTEGER' 'PROCEDURE'
'BOOLEAN' 'ARRAY'	'LABEL'
'BOOLEAN' 'PROCEDURE'	'PROCEDURE'
'EXTENDED REAL'	'REAL'
'EXTENDED REAL' 'ARRAY'	'REAL' 'ARRAY'
'EXTENDED REAL' 'PROCEDURE'	'REAL' 'PROCEDURE'
'INTEGER'	'STRING'
	'SWITCH'

5. The specifiers indicate for the parameters in their "list" what form the corresponding actual parameters should take. (Note: 'INTEGER', 'REAL', and 'EXTENDED REAL' may be used interchangeably and the proper transformations will take place automatically.)

6. A formal parameter may appear in no more than one "list." However, a formal parameter need not appear in a "list," except for switches which must be specified.

7. Statement s may be

   a. a simple statement

   b. a compound statement

   c. a block

8. Identifiers which are not formal parameters may appear in s if either of the following conditions exists:

   a. s is a block and the identifiers are declared at the beginning of this block.

   b. the identifiers are declared in the block in which the procedure declaration appears.

9. Statement s always acts like a block insofar as the scope of its identifiers is concerned; i.e., a label appearing in s is not defined outside the procedure declaration.

10. The procedure specified may be executed anywhere in the block in which the declaration appears by writing a procedure statement containing the procedure identifier and the actual parameters.

'PROCEDURE'

Example

```
'PROCEDURE' SQUAREROOT (X,E,S);
'REAL' X, E, S'
'BEGIN' 'REAL' SA;
   'IF' X 'LS' 0 'THEN'
   'BEGIN' S←-1; 'GO TO' B 'END';
   SA←1;
   A: S←(SA+X/SA)/2;
   'IF' ABS(SA-S) 'GR' E 'THEN'
      'BEGIN' SA←S; 'GO TO' A 'END';
B: 'END'
```

This procedure computes the square root. Input formal parameters are: number X whose square root is wanted and accuracy E. Output formal parameter is square root S of X.

'PROCEDURE'

'PROCEDURE' DECLARATION, VALUE AND SPECIFICATION PART

To define a statement or series of statements as being associated with a procedure identifier; to provide a means by which a procedure may be executed any number of times in the course of a program although the steps of the procedure appear only once; to specify which formal parameters are replaced by the value of the corresponding actual parameters; to specify the kinds of quantities actual parameters may represent.

Form

'PROCEDURE' name (a_1 t a_2 t ... t a_n)

'VALUE' list;

sp list; sp list;...; sp list; s

name: procedure identifier

$a_1, a_2, \ldots, a_n$: formal parameters

t: separator

sp: specifier

s: statement

list: formal parameters separated by commas

Rules

1. Each separator t may be either "," or ")b:(" where b represents any sequence of letters. The function of b is only descriptive; i.e., it may be used as comments to describe actual parameters. b has no operational significance.

2. The formal parameters may be any of the following:

   a. variable

   b. array identifier

   c. label

   d. switch identifier

   e. procedure identifier

3. The formal parameters usually appear somewhere in statement s. They are replaced at the time the procedure is called upon by the actual parameters of the procedure statement.

4. However those formal parameters which are listed in the 'VALUE' part of the declaration are assigned the current values of the corresponding actual parameters before statement s is executed. The order of assignment is from left to right according to the order of appearance in the formal parameter list.

5. The specifier may be any of the following:

'ARRAY'	'INTEGER' 'ARRAY'
'BOOLEAN'	'INTEGER' 'PROCEDURE'
'BOOLEAN' 'ARRAY'	'LABEL'
'BOOLEAN' 'PROCEDURE'	'PROCEDURE'
'EXTENDED REAL'	'REAL'
'EXTENDED REAL' 'ARRAY'	'REAL' 'ARRAY'
'EXTENDED REAL' 'PROCEDURE'	'REAL' 'PROCEDURE'
'INTEGER'	'STRING'
	'SWITCH'

6. The specifiers indicate, for the parameters in their list, what form the corresponding actual parameters should take. (Note: 'INTEGER', 'REAL' and 'EXTENDED REAL' may be used interchangeably and the proper transformations will be made automatically.)

7. A formal parameter may appear in no more than one specification list. However, a formal parameter need not appear in a list, except for switches which must be specified.

8. A formal parameter appearing in the 'VALUE' list must also appear in one of the specification lists.

9. Statement s may be:

   a. a simple statement

   b. a compound statement

   c. a block

10. Identifiers which are not formal parameters may appear in s if either of the following conditions exists:

    a. s is a block and the identifiers are declared at the beginning of this block.

    b. the identifiers are declared in the block in which the procedure declaration appears.

11. Statement s always acts like a block insofar as the scope of its identifiers is concerned; i.e., a label appearing in s is not defined outside the procedure declaration.

12. The procedure specified may be executed anywhere in the block in which the declaration appears by writing a procedure statement containing the procedure identifier and the actual parameters.

Example

```
'PROCEDURE' TOT (T,K,L,M,U);
'VALUE' L,M; 'INTEGER' L,M;
'BEGIN'
   T ← 0;
   'FOR' K ← L 'STEP' 1 'UNTIL' M 'DO'
   T ← T+U
'END'
```

This procedure computes the sum of values of a function U between the limits of summation L and M. The function U may depend on the summation index K. The sum is generated in formal parameter T.

'PROCEDURE'

'PROCEDURE' DECLARATION, FUNCTION DEFINITION

To define a statement or series of statements associated with a specific procedure identifier as being a function; to provide a means by which the appearance of the procedure identifier will cause the function to be performed and a value to be given to the identifier although the steps of the function appear only once.

Form

(1) type 'PROCEDURE' name; s

(2) type 'PROCEDURE' name (a_1 t a_2 t ... t a_n);s

(3) type 'PROCEDURE' name (a_1 t a_2 t ... t a_n);
sp list; sp list;...; sp list; s

(4) type 'PROCEDURE' name (a_1 t a_2 t ... t a_n);
'VALUE' list;

sp list; sp list;...; sp list; s

type: type word

name: procedure identifier

$a_1, a_2, \ldots, a_n$: formal parameters

t: separator

sp: specifier

s: statement

list: formal parameters separated by commas

Rules

1. A procedure declaration may have no parameters, as shown in Form (1).

2. When there are parameters (Forms (2), (3), and (4)), each separator t may be either "," or ")b:(" where b represents any sequence of letters. The function of b is only descriptive; i.e., it may be used as comments to describe actual parameters. b has no operational significance.

3. The type word may be any of the following:

   a. 'INTEGER'

   b. 'BOOLEAN'

   c. 'REAL'

   d. 'EXTENDED REAL'

   The type word identifies the type of the procedure identifier.

4. At some point in the procedure body, i.e., in statement s, the procedure identifier must appear on the left side of an assignment statement. When this statement is executed, the function receives a value, and it is this value which is used when the procedure identifier appears in an expression. The function receives a value according to the type specified by the type word.

5. The procedure identifier may appear on the left side of any number of assignment statements. It is the last one to be executed from which the function receives its value.

6. The formal parameters may be any of the following:

   a. variable

   b. array identifier

   c. label

   d. switch identifier

   e. procedure identifier

7. The formal parameters usually appear in statement s. They are replaced at the time the procedure is called upon by the actual parameters of the function call.

8. There may or may not be a 'VALUE' declaration in a function definition. If there is, the rules which apply are the same for all procedure declarations.

9. The specifiers which may be included, and the rules which apply are the same for all procedure declarations.

10. Statement s may be

    a. a simple statement

    b. a compound statement

    c. a block

11. Identifiers which are not formal parameters may appear in s if either of the following conditions exist:

    a. s is a block and the identifiers are declared at the beginning of this block.

b. the identifiers are declared in the block in which the procedure declaration appears.

12. Statement s always acts like a block insofar as the scope of its identifiers is concerned, i.e., a label appearing in s is not defined outside the procedure declaration.

13. The function which this declaration defines may be executed anywhere in the block in which this declaration appears by writing in an arithmetic or Boolean expression, the procedure identifier, and the actual parameters, if any.

Examples

1.
```
'REAL' 'PROCEDURE' ADD (K,L,M,U);
'BEGIN' 'REAL' W;
   W←0;
   'FOR' K←L 'STEP' 1 'UNTIL'
   M 'DO'
      W←W+U;
   ADD←W
'END'
```

This function computes the sum of values of a function U between the limits of summation L and M. The function U may depend on the summation index K. Upon exit from the function, the sum is contained in ADD which is of type 'REAL'.

2.
```
'INTEGER' 'PROCEDURE' FACT(X);
'IF' X 'EQ' 1 'THEN' FACT←1 'ELSE'
   FACT←X*FACT(X-1)
```

This is an example of a recursive procedure declaration. Execution of FACT(2) causes FACT(1) to be executed because of the statement FACT←2* FACT(1). Then FACT will have the value 2*1. Execution of FACT(3) causes FACT to have the value 3*2*1. If this procedure is called n times, FACT will have the value n factorial.

'PROCEDURE' DECLARATION, SEPARATELY COMPILED

To provide a technique for communicating with separately compiled procedures.

Form

(1) 'CODE'

(2) 'CODE' 'BEGIN' $d_1;d_2;\ldots;d_n$ 'END'

$d_1,d_2,\ldots,d_n$: code declarations

Rules

1. Form (1) or Form (2) above are to be used in a procedure declaration in place of statement s when it is desired to write a procedure outside an ALGOL program. The procedure may be written either as a separately compiled ALGOL program or as a procedure compiled in some other language (e.g., GMAP).

2. Each d_i may have any one of the following forms:

   a. 'OWN' type 'ARRAY' $a_1,a_2,\ldots,a_n$

      where type and $a_1,a_2,\ldots,a_n$ have the same meaning as described under Array declaration, 'OWN'. This code declaration declares 'OWN' arrays whose storage will be reserved in the declaring program but whose identifiers will be valid only in the separately compiled procedure.

   b. 'OWN' type $v_1,v_2,\ldots,v_n$

      where type and $v_1,v_2,\ldots,v_n$ have the same meaning as described under Type declaration, 'OWN'. This code declaration declares 'OWN' variables whose storage will be reserved with the declaring program but whose identifiers will be valid only in the separately compiled procedure.

   c. 'NONLOCAL' $a_1,a_2,\ldots,a_n$

      where $a_1,a_2,\ldots,a_n$ may be any of the following:

      1) variable

      2) procedure identifier

      3) array identifier

      4) switch identifier

      5) label

This code declaration makes the specified identifiers of the declaring procedure available to the separately compiled procedure.

3. The procedure identifier of a separately compiled procedure and all the identifiers specified in a, b, and c above must be unique in 6 characters. (A character is either a letter or a digit.)

4. For all procedures defined as 'CODE', a SYMREF will be produced in the declaring program.

5. SYMDEFS will be produced for all 'OWN' variables and arrays. In the case of an 'OWN' variable, the SYMDEF will point to the storage location for the variable; in the case of an 'OWN' array it will point to the first word of the alpha vector for the array.

6. There will be a SYMDEF associated with each entry in a 'NONLOCAL' list.

   a. For a procedure identifier, the SYMDEF will point to the entry location of the procedure.

   b. For a switch identifier, the SYMDEF will point to the entry location for the body of code which evaluates the switch.

   c. For a variable identifier, the SYMDEF will define either the absolute location or the stack relative location of the variable, depending on whether the variable is nonprocedural or procedural.

   d. For a label, the SYMDEF will point to the location of the label.

   e. For an array identifier, the SYMDEF will point to the first word in the alpha vector for the array. The pointer will be absolute or stack relative, depending on the point of definition of the array.

7. The user of a 'CODE' procedure is completely responsible for proper manipulation of the stack pointer, for setting of the available space pointer, and for correct usage of the various ALGOL constructs made available to him.

8. It is possible to remap the internal name of a separately compiled procedure into a different set of 6 or fewer characters which will be used as its SYMREF. This is accomplished with the ALGOL word 'RENAME' followed by a string containing the desired external name. This construct follows the formal parameter list and precedes the word 'CODE'. The 'RENAME' string may consist of any combination of 6 or fewer characters and/or decimal points.

Example

```
'PROCEDURE' INPUT 0 (a, string); 'RENAME'".A0IPT\; 'CODE'
```

'SWITCH' DECLARATION

To set up a list of statement labels and/or switch designators which will be referred to by subsequent 'GO TO' statements.

Form

'SWITCH' sw ← $d_1, d_2, \ldots, d_n$

sw: switch identifier

$d_1, d_2, \ldots, d_n$: designational expression(s)

Rules

1. This statement defines the switch identifier sw as being associated with a list of designational expressions separated by commas.

2. A designational expression (d_i) is any one of the following:

   a. a statement label

   b. a switch designator. This has the form sw[a], where sw represents a switch identifier and a represents an arithmetic expression.

   c. a conditional designator. This has the form

      'IF' b 'THEN' c 'ELSE' d

      where

      b represents a Boolean expression;

      c may be either a statement label, a switch designator or a conditional designator enclosed within parentheses;

      d may be either a statement label, a switch designator, or a conditional designator (not necessarily enclosed within parentheses).

3. Each designational expression is identifier by a positive integer - the leftmost with 1, the next with 2, etc.

4. When a 'GO TO' statement involving a switch designator is encountered in the program, the subscript of the switch designator is given an integral value. It is this value which determines which element of the list is referenced.

5. If the list item referenced is a conditional designator, the 'IF' clauses are evaluated until a designational expression involving only a label or a switch designator is reached.

6. If the list element referenced is a label, it specifies directly the next statement to be executed.

7. If the element is a switch designator, it in turn references another 'SWITCH' declaration. The subscript of the switch designator is evaluated to locate the correct list element of the new 'SWITCH' declaration.

8. This process may be repeated through any number of 'SWITCH' declarations until reference is made directly to a statement label.

9. Each time an element in the list of a 'SWITCH' declaration is referenced, any expressions the element may contain are re-evaluated.

Example

```
'SWITCH' BA← PA, 'IF' S 'THEN' PB 'ELSE' PC, AC[X]
```

This switch may be called by a statement such as 'GO TO' BA[D] which operates as follows: If D has the value 1, operation is equivalent to operation of 'GO TO' PA, where PA is a statement label. If D has the value 2, operation is equivalent to operation of 'GO TO' 'IF' S 'THEN' PB 'ELSE' PC, where S is a Boolean expression and PB and PC are statement labels. If D has the value 3, operation is equivalent to operation of 'GO TO' AC[X] where AC is a switch identifier and X is an arithmetic expression. If D has any other value, the statement following the 'GO TO' is executed next.

TYPE

TYPE DECLARATIONS

To specify which variables represent integer, real, extended real, or Boolean quantities.

Form

type $v_1, v_2, \ldots, v_n$

type: type word

$v_1, v_2, \ldots, v_n$: variable(s)

Rules

1. The type word may be one of the following:

   'REAL', 'EXTENDED REAL', 'INTEGER', or 'BOOLEAN'.

   The type word specifies the type of the variables $v_1, v_2, \ldots, v_n$.

2. Each variable used in a program must be declared in a type declaration.

3. No variable may appear in more than one type declaration in a single block.

4. The type declaration is valid only for the block in which the declaration appears. Outside this block the identifiers may be used for other purposes.

5. The type declaration is valid for any blocks contained within the block containing the type declaration. However, variables may be redeclared in sub-blocks, in which case the previous declaration is superseded.

6. When exit is made from a block (by 'END' or by a 'GO TO' statement) all identifiers which were declared for the block are undefined.

Example

```
'BEGIN' 'INTEGER' P,Q; 'INTEGER' 'ARRAY' S[1:5];
   P ← 3; Q ← 2;
   'BEGIN' 'REAL' P,R;
      R ← Q;
      P ← 1;
      S[1] ← P;
      S[2] ← Q;
      S[3] ← R
   'END';
   S[4] ← P;
   S[5] ← Q
'END'
```

These statements assign the numbers 1,2,2,3,2 in this order to elements of the array S.

TYPE, 'OWN'

To specify which variables represent integer, real, extended real, or Boolean quantities; to provide a means for retaining previous values of certain variables upon re-entry into a block.

Form

'OWN' type $v_1, v_2, \ldots, v_n$

type: type word

$v_1, v_2, \ldots, v_n$: variable(s)

Rules

1. The type word may be one of the following:

   'REAL', 'EXTENDED REAL', 'INTEGER', or 'BOOLEAN'.

   The type word specifies the type of the variables $v_1, v_2, \ldots, v_n$.

2. Each variable used in a program must appear in a type declaration.

3. No variable may appear in more than one type declaration in a single block.

4. Only variables whose values are to be preserved for possible re-entry into a block should be specified by an 'OWN' type declaration. All other variables should be declared in a regular type declaration.

5. The variable identifiers declared in any type declaration are defined only for the block in which they appear. Outside the block the identifiers may be used for other purposes.

6. When an exit is made from a block (by 'END' or by a 'GO TO' statement) the identifiers are inaccessible although their values have been saved.

Example

```
B:  'BEGIN' 'REAL' C; 'OWN' 'REAL' D;
      'IF' A 'EQ' 6 'THEN'
      'BEGIN'
          C ← 7;
          D ← 8;
          A ← 9;
          'GO TO' E
      'END';
      A ← D-2
    'END';
E:  'IF' A 'NQ' 6 'THEN' 'GO TO' B
```

During the first execution of block B, 7 is assigned to C, 8 is assigned to D and 9 is assigned to A. Execution of the conditional statement labelled E causes block B to be executed again. During this execution, A is set to 6 because the previous value of 'OWN' variable D is saved. However, variable C could not be used in this way because not being 'OWN', its value is not saved.

'LINK' DECLARATION

To permit separate compilation of parts of a program while retaining the "environment" of the entire program; to allow different parts of a program to occupy the same area of storage at object execution time.

Form

```
'LINK' s;

s: string
```

Rules

1. Links may be blocks, procedures, or function procedures.

2. Links may have sublinks which may have sublinks, etc.

3. Links must be separately compiled.

4. All identifiers which would have been known to a link, had its coding appeared in the program, will be known to the link without special indication.

   Note. Currently the one exception is a code-body procedure. This must be redeclared in the link.

5. The declaration for a block link is:

   'BEGIN' 'LINK' "NAME\; 'END'

   The declaration for a procedure link is:

   'PROCEDURE' ABC (X,Y); 'LINK' "NAME\;

6. The string following the declarator 'LINK' may contain up to five alphanumeric characters. (A period may also be used.)

7. The name following 'LINK' must be unique within a program. Here a program means a main program plus all links and sublinks pertaining to it.

8. For a link compilation, the first basic symbol must be 'LINK'. This must be followed by the name string which appeared in the declaring program.

   For a block link compilation:

   'LINK' "NAME\; 'BEGIN' declarations and statements
   'END'

'LINK'

For a procedural link compilation:

'LINK' "NAME\; 'PROCEDURE'

9. 'OWN' declarations are allocated storage in the declaring program. It is the programmer's responsibility to ensure that 'OWN' values, which are to be saved, are not overlayed by another link.

10. No special calling sequence is necessary to invoke a block link or a procedure link.

11. For programs which are links or which contain link declarations, a link file (file code LF) must be provided with a GCOS file control card. This file may be assigned to magnetic tape, linked disk, or linked drum.

12. When compiling a link the environment of the main program must be known to the link. This is provided by the link file LF. This means that the link file which was prepared by a program in which a link was declared must be used when that link is compiled.

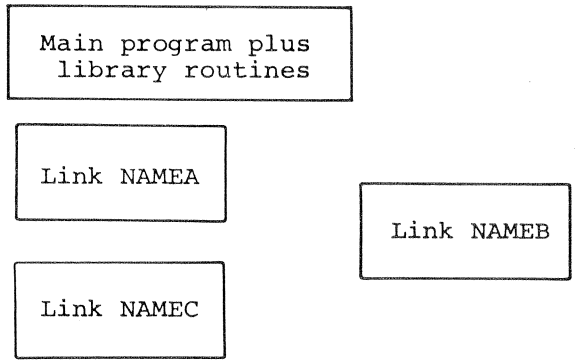
13. The name following 'LINK' is the name which will appear on the $ LINK control card.

14. A $ ENTRY control card must also be provided for each link. The name on this card must be the 'LINK' name preceded by a period.

Example

```
$  OPTION ALGOL
$  ALGOL
$  DISC   LF,X1S,5L
   'BEGIN'
        .
        .
        'BEGIN' 'LINK' "NAMEA\; 'END';
        'BEGIN' 'LINK' "NAMEB\; 'END';
        .
        .
   'END'
$  LINK   NAMEA
$  ALGOL
$  DISC   LF,X1S,5L
   'LINK' "NAMEA\;
   'BEGIN'
        .
        .
        'PROCEDURE' ABC; 'LINK' "NAMEC\;
        .
        .
   'END'
$  ENTRY.NAMEA
$  LINK  NAMEC
$  ALGOL
$  DISC   LF,X1S,5L
   'LINK' "NAMEC\;
   'PROCEDURE' ABC; 'BEGIN'....'END'
$  ENTRY.NAMEC
$  LINK  NAMEB,NAMEA
$  ALGOL
$  DISC   LF,X1R,5L
   'LINK' "NAMEB\;
   'BEGIN'
        .
        .
        .
        .
   'END'
$  ENTRY.NAMEB
$  EXECUTE
$  TAPE H*,X1R
```

These would appear in memory at execution time as



Link NAMEC is a sublink of link NAMEA. When link NAMEB is called, it would overlay memory occupied by links NAMEA and NAMEC.

SECTION VII

COMPOUND STATEMENT AND BLOCK

DESCRIPTIVE FORMAT

The description of the compound statement and block is presented in this Section. Each description starts on a new page, with the format of its descriptive material given as shown below:

Descriptive name

(A brief statement of the purpose of the statement)

Form

(Form of the statement)
(Definition of symbols used in the form line)

Rules

(A list of rules governing the correct usage of the statement; includes restrictions, suggestions, etc.)

Examples

(A list of examples illustrating the use of the statement)

COMPOUND STATEMENT

COMPOUND STATEMENT

To permit a series of statements to be joined together in such a way as to act as a unit.

Form

'BEGIN' $s_1;s_2;\ldots;s_n$ 'END'

$s_1,s_2,\ldots,s_n$: statements

Rules

1. A compound statement may have a label and may contain any number of statements (s_i).
2. Each statement $s_1,s_2,\ldots,s_n$ may be
   a. a simple statement
   b. a compound statement
   c. a block
3. Each statement may have a label.
4. A 'GO TO' statement may transfer control to a statement within a compound statement.

Examples

1.
```
I ← 1;
T: 'IF' I 'LQ' 19 'THEN'
'BEGIN'
   A[I] ← I;
   I ← I+1;
   'GO TO' T
'END'
```

These statements assign the numbers one to ten to elements of the array A. This example contains a compound statement as the true branch of a conditional statement.

COMPOUND STATEMENT

2.
```
'FOR' I←1 'STEP' 1 'UNTIL' 10 'DO'
   'BEGIN'
   'FOR' J←1 'STEP' 1 'UNTIL' 10 'DO'
      'BEGIN'
      'IF' I 'EQ' J 'THEN'
         'BEGIN'
         B[I,J] ←1; 'GO TO' S
         'END';
      B[I,J] ← 0;
   S: 'END'
   'END'
```

These statements generate a ten by ten unit matrix in the array B. Each 'FOR' statement has a compound statement as its object. Also, the true branch of the 'IF' statement is a compound statement.

BLOCK

To permit statements and declarations to be grouped together in such a way as to be independent of other parts of a program. This permits labels and identifiers to be used in different sections of a program without conflicts.

Form

'BEGIN' $d_1;d_2;\ldots;d_n;s_1;s_2;\ldots;s_n$ 'END'

$d_1,d_2,\ldots,d_n$: declarations

$s_1,s_2,\ldots,s_n$: statements

Rules

1. A block may have a label, and may contain any number of declarations and statements.
2. Each statement $s_1,s_2,\ldots,s_n$ may be
   a. a simple statement
   b. a compound statement
   c. a block
3. Each statement may have a label.
4. When a block is entered through 'BEGIN', the identifiers which are declared for the block are newly defined and lose any significance they may have had prior to entry.
5. All labels within a block are local to the block and may not be referred to from outside.
6. When exit is made from a block, all identifiers which were declared for the block are undefined and may be used for other purposes, including those declared as 'OWN'.
7. If a declaration is prefaced with 'OWN', the identifiers so defined will retain their previous values upon re-entry into the block. If 'OWN' is not specified, the values will be lost when exit is made from the block and will be undefined upon re-entry.
8. All identifiers used in a program must be declared in one of the blocks comprising the program. No identifier may be declared more than once in a single block.
9. If blocks are nested, a statement label has meaning only in the smallest block containing that statement.

BLOCK

Example

```
'BEGIN' 'REAL' X,Y; 'ARRAY' A[1:5];
   X ← 1; Y ← 2;
   'BEGIN' 'REAL' X,Z;
      Z ← Y;
      X ← 3;
      A[1] ← X;
      A[2] ← Y;
      A[3] ← Z
   'END'
   A[4] ← X;
   A[5] ← Y
'END'
```

These statements assign the numbers 3,2,2,1,2 in this order to elements of the array A.

SECTION VIII

INPUT/OUTPUT

INPUT/OUTPUT PROCEDURES

The ALGOL language itself provides no input/output statements. However, the ALGOL compiler contains within it a number of procedures which handle the I/O. All a programmer need do is to call the existing procedures using an ALGOL procedure statement, and through the procedure parameters, transmit the information required for the input and/or output process.

The procedure identifiers used by ALGOL are reserved and act as though declared in a block enclosing the program. If a programmer redeclares one of these identifiers in his program his declaration supersedes the standard definition. The procedures provided are listed below:

- Procedures pertaining to the layout of the I/O information on the external device:

  BAD DATA
  FORMAT
  FORMAT n (n=0,1,2,...,9)
  HEND
  HLIM
  NO DATA
  TABULATION
  VEND
  VLIM

- Procedures dealing with the actual transmission of data:

  INLIST
  INPUT n (n=0,1,2,...,9)
  OUTLIST
  OUTPUT n (n=0,1,2,...,9)

- Procedure allowing fine control over the input and output processes:

  SYSPARAM

- Primitive procedures:

  INSYMBOL
  LENGTH
  NAME
  OUTSYMBOL
  STRING ELEMENT
  TYPE

- List procedure:

  A user-declared procedure providing a list of the data items to be transmitted.

Each procedure is discussed in detail below, and the form of the procedure call is given.

INPUT/OUTPUT DEVICES

The procedures to be described in this section deal with the appearance of the data on an input or output device. All of the procedures describe a printed page. However, the concepts may be generalized to include any external device.

Listed below are the physical characteristics of the I/O devices. The number of characters per line is referred to as P. The number of lines per page is referred to as P'.

Device	P (characters)	P' (lines)
Line Printer	120	55
Card Reader (binary)	160	no limit
Card Reader (decimal)	80	no limit
Card Punch (binary)	160	no limit
Card Punch (decimal)	80	no limit
Magnetic Tape, Disk, Drum	120	no limit

These device characteristics may be altered where applicable (e.g., number of characters per line for magnetic tape may be reduced) by using the procedure SYSPARAM.

DESTINATION CODE

The Series 600/6000 ALGOL system uses a symbolic destination code which indicates the medium to which the information is intended to go (final destination) or from which it is to come. Destination codes are required because information is often stored temporarily on an intermediate device such as magnetic tape or disk. The destination code indicates the format of the data on the external device. For example, a common case is when information which will eventually be printed is temporarily stored on tape. In this case the destination code indicates that the data is to be formatted as if it were going to the printer, in spite of the fact that it is going to magnetic tape. Another case is when a card image is read from disk. In this case the destination code indicates that the data is to be read in the same format as when reading directly from the card reader.

The destination code defines the type of logical device to be associated with a file, independent of the physical device. In this way the system limits (that is, P and P') for a file are defined. When the destination code is absent, the logical device will be the same as the physical device.

The destination code is specified on a $ FFILE control card by the option DSTCOD/(XXX), where XXX may be any of the following:

Mnemonic	Definition
MTAPE	magnetic tape
DISC	disk
DRUM	drum
BCRDR	card reader, binary mode
DCRDR	card reader, decimal mode
BCPNCH	card punch, binary mode
DCPNCH	card punch, decimal mode
PRNTR	printer
PTMODS	5/6 channel paper tape
PTMODD	7/8 channel paper tape

The layout procedures are used to describe nonstandard operations which are to take place during input and output. The procedures need not be called, in which case certain standard operations (described with each procedure) will be in effect. The technique for using the layout procedures is as follows:

The programmer declares a setup procedure containing any or all of the eight layout procedures (FORMAT, HLIM, VLIM, HEND, VEND, NO DATA, TABULATION, BAD DATA). At some point in the program there is a call to an I/O transmission procedure which has as one of its parameters the procedure identifier of this setup procedure. At the time the I/O procedure is called it causes the setup procedure to be executed, thus establishing the nonstandard operations. Each time a new I/O transmission is called, the standard layout operations will be resumed until changed by a new setup procedure.

BAD DATA

To indicate the procedure which is to be called when a request is made for an item to be transmitted, and the item is incompatible with the format character.

Form

BAD DATA (p)

p: procedure identifier

Rules

1. This procedure applies only to input.
2. If the referenced field is not compatible, control will be transferred to procedure p.
3. If BAD DATA is not used and the condition described in Rule 2 arises, control will be transferred to the end of the program as though a dummy label had been placed just before the final 'END'.

Examples

1. BAD DATA (CHECK)

   The procedure CHECK is used when incorrect data appears on the input device.

2. 'BEGIN' 'PROCEDURE' REDO; OUTLIST (6,LAY,LIST);
   ... BAD DATA (REDO);... 'END'

   When an incompatibility occurs, control goes to procedure REDO which outputs an error message.

FORMAT

To describe the form in which data appears on the input device or is to appear on the output device.

Form

FORMAT (string)

string: a string with a special form

Rules

1. The format string is composed of a series of items separated by commas.

2. The string is interpreted from left to right in conjunction with a list of data items which are to be transmitted.

3. These data items usually appear in a separate procedure called a list procedure.

4. An item in the format string may describe a number, a string, or a Boolean quantity, or it may simply cause a title to be written or page alignment to take place.

5. All the format items listed above constitute a format string.

6. Any format item or any group of format items can be repeated any number of times by enclosing in parentheses those items to be repeated and preceding the parentheses by an integer n indicating the number of repetitions desired; i.e., 3(2Z.D) causes 3 decimal numbers to be transmitted. If no integer precedes the parentheses an infinite number of repetitions is indicated.

Number Formats

1. Integers. This format item consists of a series of Z's, a series of D's, or a series of Z's followed by D's, each corresponding to a digit position of the number, and an optional sign.

The letter D is used to indicate a digit which is always to be printed. (For example, 385 when written with format DDDD will appear externally as 0385.)

The letter Z is used to indicate that the corresponding digit is to be suppressed if it is a leading zero. In this case, a zero digit will be replaced by a blank space when all the digits to its left are zeros. (For example, 21 when written with format ZZZ will appear externally as ø21.)

A series of Z's or D's may be written in a shorthand notation as follows: nZ or nD (where n is an integer) is equivalent to ZZZ...Z or DDD...D (n times). (For example, 3Z and ZZZ are equivalent. 4D and DDDD are equivalent.)

An optional sign may precede or follow the Z's and D's of a number format. If no sign appears, the number is assumed to be positive. Note: If a negative number is output with no sign position, the first digit position will print as ø,A,B,...,I representing the digits 0,1,2,...,9 respectively. If a plus sign appears, the correct sign of the number appears on the external medium. If a minus sign appears, positive numbers will be unsigned and negative numbers will have a minus sign on the external medium.

If a preceding sign is to appear externally with a number which has had leading zeros suppressed, the sign will be placed immediately to the left of the first non-zero digit.

The total number of positions which an integer occupies on the external medium is the sum of the Z's and D's (plus one if the optional sign appears). If the field width is insufficient to hold the complete number, the complete number will be transmitted; but the rest of the line may be moved to the right.

Examples of integer formats:

If +XXDDD is used with 2176, it appears as ø+2176.
If -ZZZDD is used with 3, it appears as øøøø03.
If -DDDD is used with -45, it appears as -0045.
If ZZZ is used with 0, it appears as øøø.
If ZZD is used with 0, it appears as øø0.
If 2Z4D+ is used with 390, it appears as øø0390+.

2. Decimal Numbers. This format item consists of Z's and/or D's each corresponding to a digit position, a period (.) or the letter V to indicate the position of the decimal point, and an optional sign.

The letter Z has the same function it did for integers and it may appear only to the left of the decimal point.

The letter D may appear on both sides of the point and has the same function as for integers.

If a . is used to indicate the decimal point position, it will appear on the external medium in that position. If the letter V is used it merely indicates where the decimal point should be, but no space is used on the external medium.

The sign part functions as it did for integers.

The total number of positions which a decimal number occupies on the external medium is the sum of the Z's and D's plus one for the sign, plus one if the point is indicated by a . in the format. If the field width is insufficient to hold the complete number, the complete number will be transmitted; but the rest of the line may be moved to the right.

Examples of decimal numbers:

If ZZDD.DD is used with 146.776, it appears as ∅146.78.
If -3D.D is used with 1.2, it appears as ∅001.2.
If +3Z.3D is used with .0042, it appears as ∅∅∅+.004.
If -ZZDVD is used with -142.78, it appears as -1428.
If ZZ4D.DD- is used with -3394.7, it appears as ∅∅3394.70-.
If ZZD is used with 29.756, it appears as ∅30.
If .3D- is used with -.0254, it appears as .025-.

3. Decimal Numbers with Exponent. This format item is the same as that for a decimal number with the addition of an exponent part to indicate the power of ten to which the number must be raised to give the true decimal number.

The exponent part consists of an apostrophe to separate it from the decimal number followed by an optional sign, a series of Z's, and/or a series of D's.

The letter E will appear on the external medium in the proper position to separate the decimal number and its exponent.

The rules for the exponent part are the same as those for integers.

A number using this format will appear externally with its leading digit not zero. The exponent is adjusted accordingly. If the number is zero the exponent is also set to zero.

If a nonzero number has a zero exponent which is specified by Z's the apostrophe and the exponent sign are also suppressed.

The total number of positions needed on the external medium is the sum of all the Z's and D's plus one for the sign, plus one for the exponent sign, plus one for the apostrophe plus one for the decimal point (if the decimal point is specified).

Examples of decimal numbers with exponents:

If 3D.DD'+DD is used with 3075.2, it appears as 307.52'+01.
If D.DD'-ZZ is used with 7.1, it appears as 7.10∅∅∅.
If ZZD'+ZD is used with .021758, it appears as 218'∅-4.
If DD'ZZ is used with 35.649, it appears as 36∅∅.
If .3D'+2D is used with 917.2, it appears as :917'+03.
If .DD'-ZZZ is used with .000312, it appears as :31'∅∅-3.

4. Octal Numbers. The form of this item is nO or OO...O (n times) where n is an integer which specifies the number of digits in the octal field.

The octal format item may only be used for output.

For output of real and extended real numbers, if n < 12, the leftmost n digits will be transmitted; if n ≥ 12, twelve digits will be tranmitted, followed by n - 12 blanks.

For output of integer and Boolean values, if n < 12, the rightmost n digits will be transmitted. If n ≥ 12, twelve digits will be transmitted, followed by n - 12 blanks.

Examples of octal numbers:

If 5O is used with 447521767511 on output, it appears as 44752.
If 14O is used with 712342165134 on output, it appears as 7123421651340Ø.

Truncation for Number Formats

The integer or decimal number formats described above may be followed by the letter T to indicate that the output should be truncated instead of rounded. Rounding occurs when truncation is not specified.

Examples of truncation:

If -2Z3D.2DT is used with -12.719, it appears as ØØ-012.71.
If 3ZDT+ is used with 145.6, it appears as Ø145+.
If -Z.DT'+ZZ is used with .-12537, it appears as Ø1.2'Ø-2.

Insertions in Number Formats

All of the number formats may have either blanks or strings inserted anywhere within the format item. The insertion will appear on the external medium.

A blank is denoted by the letter B. If more than one blank is desired it may be expressed by a series of B's or by the short hand notation nB (n is an integer specifying the number of blanks). 3B is equivalent to BBB.

A string which is to be inserted must be enclosed in string quotes (i.e. "string\). If the string is to be repeated it may appear as n"string\ where n is an integer specifying the number of times the string is to appear. The information in the string (not including the outermost quotes) is inserted in the corresponding place in the number.

Examples of insertions:

If D2B3D is used with 3972, it appears as 3ØØ972.
If "ANS=\4D is used with 271, it appears as ANS=0271.
If "INTEGERØPART\-4ZVB"FRACTION\B2D is used with -195.7634, it appears as INTEGERØPARTØ-195ØFRACTIONØ76.
If 2ZB2D.DBT'+DD is used with 44865.5, it appears as 44Ø86.5Ø'+01.
If "OCTALØ\5O is used with 112233445566, it appears as OCTALØ11223.

Numbers for Input

Numbers which are input using the above format codes should, in general, appear the same as those which are output.

However, there are fewer restrictions on the form of input numbers.

1. Leading zeros may appear even if Z's are used in the format code. Leading blanks may appear even if D's are used.

2. If insertion strings or blanks are used in the format code the corresponding number of characters on the input device are skipped.

3. If a sign is specified at the left in the format code it may appear in any Z or D positions on the input device as long as it is to the left of the first digit. If the sign is specified at the right, it must appear exactly where it is indicated.

4. If a period is required, it must appear exactly where it is indicated.

String Format

This format item is used to output string quantities. It may not be used for input. Alpha format must be used instead.

The form of this item is nS or SS...S(n times), where n is an integer which indicates the number of symbols in the string.

1. If the actual string is longer than the number of S's indicated, only the leftmost symbols are transmitted.

2. If the string is shorter, blank symbols are added to the right of the string.

3. Examples of string format:

   If S is used with "A\, it appears as A.
   If 6S is used with "TOTALS\, it appears as TOTALS.
   If SSS is used with "ABC\, it appears as ABC.
   If 4S is used with "PROGRAM\, it appears as PROG.
   If 5S is used with "CAT\, it appears as CATØØ.

Insertions in String Format

Blanks or strings may be inserted in the S format. The rules are the same as described for number formats.

Examples:

If B3SBB2S is used with "12345\, it appears as ∅123∅∅45.
If 2S"= 3SB is used with "TlANS\, it appears as Tl=ANS∅.

Alpha Format

This format item is used to transmit ALGOL basic symbols. (See "Introduction", Section I for a list of basic symbols.)

1. The form of this item is the letter A.

2. The appearance of the letter A as a format item causes transmission of a single symbol from or to the data item specified in the list procedure.

3. The symbol will be stored as an integer.

It may be desired to work with symbols transmitted by the A format. Therefore, a function is provided which makes any ALGOL symbol type 'INTEGER' and causes the symbol to have the same value as if it had been read in using Alpha format.

The function is called EQUIV. Its argument must be an ALGOL basic symbol enclosed in string quotes; i.e., EQUIV("'BEGIN'\).

Example:

If A format is used to read an "*" into variable ALG the statement 'IF' ALG 'EQ' EQUIV ("*) 'THEN' 'GO TO' GOOD will check that "*" was in fact the symbol which was read in.

Boolean Format

This format item is used to transmit Boolean quantities. The item may consist of the letter P or the letter F. If P is used and the quantity is true, the number 1 is transmitted; if false, 0 is transmitted.

On input if F is used, the next seven characters are examined. If they are 'TRUE', then the value of true is transmitted; if they are 'FALSE', the value of false is transmitted. On output if F is used and the value is true, then the seven characters 'TRUE'∅ are transmitted; if the value is false, then 'FALSE' is transmitted.

Insertions in Boolean Format

Blanks or strings may be inserted in the Boolean format. The rules are the same as described for number formats and for strings.

Examples:

If BBPB is used with a Boolean variable whose value is 'TRUE', it appears as ØØ1Ø.

If "THEØRELATIONØIS\BF is used with a Boolean variable whose value is 'FALSE', it appears as THEØRELATIONØISØ'FALSE'.

Standard Format

A number may be transmitted for input or output without specifying in a format item the exact form the number is to take. The number appears on the I/O device in "standard format."

If the letter N appears as a format item, it specifies that a number with standard format is to be transmitted.

Standard format for input may be defined as follows:

1. Any number of digits in any of the forms which are acceptable to integer or decimal number formats may be input.

2. The number must be terminated by an illegal character; i.e., one not normally permitted in a number, or by k blanks where k is a system parameter initially set at one. k may be changed by calling the system procedure SYSPARAM.

Note: The input medium will be positioned to pick up the illegal character on the next read. Therefore the programmer must space over the illegal character (by using RDCHAR or SYSPARAM) if the illegal character is not to be used.

If standard format is invoked, and the first line referenced contains any legal character for a number (i.e., digit, sign, decimal point or apostrophe) the right hand margin will terminate the number. If, however, the first line contains only blank characters, the subsequent lines will be searched until a legal number character is found. At this point the right hand margin is not significant, and only an illegal character or k blanks will terminate the number.

If standard format is used for output, the type of the variable determines the format on the external device.

1. For extended real variables, the value will appear as though the decimal number format B-.16DE+DD had been used.

2. For real variables, the value will appear as though B-.8DE+DD had been used.

3. For integer variables, the value will appear as though E-8ZD had been used.

4. For Boolean variables, the value will appear as though BF had been used.

Standard format will be assumed if the end of a format string is reached while there are data items in the list procedure still to be transmitted. In this case all the remaining quantities will be transmitted with standard format.

If a list of variables is to be terminated but either

1. no reference is made to a FORMAT procedure, or

2. the format call has the form FORMAT ("\)

the items will be transmitted according to standard format.

Hollerith Format

The form of this item is nH, where n indicates the number of characters to be transmitted. On input, up to six characters of code will be read. The characters will be inserted in a word, left justified, with trailing blanks.

On output, up to six characters of code will be inserted in the output record. If n is greater than six, then only the first six characters will be used, but the medium will be spaced n characters.

Alignment Marks

These are single characters which cause specific page operations to occur. The marks and operations are:

/	go to next line
↑	go to new page
J	go to next tabulation position.

Alignment marks may appear at the left or the right of any format item. If they appear at the left of the item, the actions take place before the format operation. If they appear at the right, they take place afterwards; i.e., /3S causes a skip to a new line before the string is transmitted and a skip to a new page after.

Alignment marks may appear as separate format items simply by enclosing them in commas.

Any number of alignment marks may appear in succession, and this causes the specified action to be repeated as many times as it is indicated; i.e., causes a page to be terminated and two pages to be skipped. Also any mark may be preceded by an integer n, where n indicates the number of times the action is to be done; i.e., 4J causes a skip to the fourth tab position and is equivalent to JJJJ.

Title Format

This format item is used when it is desired to cause page alignment and/or the output of insertion strings without transmitting any ALGOL quantities. This item consists entirely of insertions and alignment marks and refers to no data items.

On input, this item causes characters to be skipped corresponding to the insertion strings and causes the desired alignment operations to be performed. On output, the insertion strings are transmitted and the alignment operations are performed.

Example of title format:

↑"SUMMARY\// indicate a new page, an insertion, a line to be terminated and a line to be skipped.

Examples

1. FORMAT("4D.2D,2Z,/P,"IS␣THE␣ANS\↑,A\)

   This format string transmits a decimal number and an integer on one line; on the next line is a Boolean quantity specified by a 0 or a 1 followed by an insertion. Then a skip is made to a new page and an ALGOL symbol is transmitted.

2. FORMAT("7S,2(5Z.D'+ZZ,F),2J\)

   A seven symbol string is transmitted followed by a decimal number, a Boolean quantity, a decimal number, a Boolean quantity. Then two tabulations occur.

3. FORMAT("(ZZ.BDT-,BBB+ZZD)\)

   A decimal number and an integer are transmitted an indeterminate number of times; i.e., until the list of data items is exhausted.

FORMAT n

To describe the form in which data appears on the input device or is to appear on the output device; to permit certain elements of the format string to be variable and to have their values calculated at the time the FORMAT procedure is called.

Form

FORMAT n(string, x_1,x_2,...,x_n)

n: integer

string: string with a special form

x_1,x_2,...,x_n: expression

Rules

1. n may be 0,1,2,...,9. This value indicates the number of x's which appear following the format string. (Note: The form FORMAT (string) as discussed previously is simply a special case of this format call in which n=0.)

2. The form of the string is the same as that discussed for the procedure call FORMAT (string), with certain additional features.

3. The string may contain the letter X in various format items. The values of the x 's which follow the format string will replace each X when the FORMAT procedure is called.

4. The letter X may appear in the format string as follows:

   a. In a Number Format:

      Any Z or D may be preceded by the letter X to indicate a variable number of repetitions of the Z or D.

      Examples:

      XZXD - variable integer size
      ZZ.XD - variable number of decimal places
      :DDD'XD - variable exponent size

   b. In an Insertion:

      The letter B may be preceded by an X to indicate a variable number of blank spaces on output or a variable number of ignored positions on input.

      Example:

      2ZXB3D.D

   c. In a String Format:

      The letter S may be preceded by the letter X to indicate a variable number of symbols in the string.

      Examples:

      XS
      BXSB4S

   d. With an Alignment Mark:

      ↑,/ or J may be preceded by the letter X to indicate a variable number of times the specified alignment action is to be taken.

      Examples:

      XJDD.DD - variable number of tabulations
      3S2BX/ - variable number of lines to be skipped

5. The X's may be used at most 9 times in a single format string. The integer n in the format call indicates the number of X's which appear in the string.

6. The $x_1,x_2,\ldots,x_n$ in the format call represent the integral values to be assigned to the X's in the string. $x_1,x_2,\ldots,x_n$ must be positive. x_1 is assigned to the first X which appears; x_2 to the second, etc.

Example

```
FORMAT 3 ("ZZXD.D,XBXS\,2,A-5,B)
```

The decimal number will be transmitted as though it had been written as ZZ2D.D. A-5 blanks will precede the string which will contain B symbols.

HEND

To specify the procedures which are to be called when the end of a line is reached during input or output; to permit special action to be taken depending on what situation causes the end-of-line condition to occur.

Form

HEND (p1,p2,p3)

p1,p2,p3: procedure identifiers

Rules

1. p1 is the name of the procedure to be called when a "/" appears in the format call. This indicates that a new line is to begin and is considered the normal case.

2. p2 is the name of the procedure to be called when a group of characters is to be transmitted or a tabulation is specified which would pass the right margin of the current line as specified by HLIM.

3. p3 is the name of the procedure to be called when a group of characters is to be transmitted or a tabulation is specified which would pass the physical end of the line due to the characteristics of the I/O device being used, but would not pass the right margin as set by HLIM. Note: This physical end is specified by standard limits set within the system or a control card to the system, and may be altered by procedure SYSPARAM.

4. If it is desired to take no special action when the end of a line is reached this procedure call may be omitted.

5. If action is desired for some but not all of the conditions, dummy procedure names may be used for those requiring no action.

Examples

1. HEND (NORM,OVER,END)

   A "/" in the format call causes control to go to Procedure NORM; if the right margin is reached control goes to OVER; if the physical end-of-line is reached control goes to END.

2. 'BEGIN'...'PROCEDURE' DUMMY;;...
   ...HEND (DUMMY,FIN,NEXT);...'END'

   Since Procedure DUMMY contains no statements, no special action will be taken when a "/" appears in the format call.

HLIM

To specify the left and right margins of the input or output lines.

Form

HLIM (left, right)

left, right: arithmetic expressions

Rules

1. The first parameter specifies the left margin.
2. The second parameter specifies the right margin.
3. There is a restriction that $1 \leq \text{left} \leq \text{right}$.
4. If this procedure call is not given, the left margin is set to one and the right margin is set to infinity.

Examples

1. HLIM (5,50)

   Left margin is 5, right margin is 50.

2. HLIM (J-4,K)

   Left margin is value of J-4, right margin is value of K.

NO DATA

To indicate the procedure which is to be called when a request is made for data on an input device but no more data remains.

Form

NO DATA (p)

p: procedure identifier

Rules

1. This procedure call applies only to input.
2. If input data is requested by a data transmission procedure when no data remains on the input device, control will be transferred to procedure p.
3. If NO DATA is not used and the condition described in Rule 2 arises, control will be transferred to the end of the program as though a dummy label had been placed just before the final 'END'.

Examples

1. NO DATA (EOF)

   The procedure EOF is used when no data exists on the input device.

2. 'BEGIN'
   'PROCEDURE' LAST; 'GOTO' FIND;
   ...
   NO DATA (LAST);...
   'END'

   When no data is found on the input device, control goes to procedure LAST which sends control to the statement labelled FIND.

TABULATION

To set the width of the tabulation field of the I/O device; to permit the skipping of a fixed number of positions whenever the alignment mark J appears in a format call.

Form

TABULATION (a)

a: arithmetic expression

Rules

1. "a" specifies the number of characters of the foreign medium which constitute the tabulation field.

2. If the left margin is at position X, the tab positions for a line are:

   X, X+a, X+2a, X+3a, ... , X+ka

   The last tab position occurs before or at the same point as the right margin as specified by HLIM, or at the physical end of the line, whichever is smaller.

3. When a "J" appears in a format call, the I/O device is spaced to the next tab position.

4. If this procedure call is not given the tabulation spacing is one.

Examples

1. TABULATION (15)

   A new tab position occurs every 15 spaces.

2. TABULATION (A↑2-B*C)

   The value of A^2-BC determines the tab spacing.

VEND

To specify the procedures which are to be called when the end of a page is reached during input or output; to permit special action to be taken depending on what situation causes the end-of-page condition to occur.

Form

```
VEND (p1,p2,p3)

p1,p2,p3: procedure identifiers
```

Rules

1. p1 is the name of the procedure to be called when a "↑" appears in the format call. This indicates that the subsequent information is to appear on a new page, and is considered the normal case.

2. p2 is the name of the procedure to be called when a group of characters is to be transmitted which would appear on the line after the one specified by VLIM as the bottom margin.

3. p3 is the name of the procedure to be called when a group of characters is to be transmitted which would pass the physical end of the page due to the characteristics of the I/O device being used, but would not pass the bottom margin set by VLIM. Note: This physical end is specified by standard limits set within the system or a control card to the system, and may be altered by procedure SYSPARAM.

4. If it is desired to take no special action when the end of a page is reached this procedure call may be omitted.

5. If action is desired for some but not all of the conditions, dummy procedure names may be used for those requiring no action.

Examples

1. VEND (NEW,PAGE 1,PAGE 2)

   Control goes to procedure NEW when a "↑" appears in the format call; to PAGE 1 when the bottom margin is reached and to PAGE 2 when the physical end of the page is reached.

2.
```
'BEGIN'...'PROCEDURE' EMPTY;;...
 ...VEND (OK,FIX,EMPTY);...'END'
```

   No action is taken if an attempt is made to write beyond the end of the page.

VLIM

To set the vertical layout of a page; to specify how many lines on a page are to be used.

Form

VLIM (top, bottom)

top, bottom: arithmetic expressions

Rules

1. The top line of the page has a value of 1, the second, 2, etc.
2. The first parameter indicates the first line to be used for transmission.
3. The second parameter indicates the last line to be used.
4. There is a restriction that 1 $\leq$ top $\leq$ bottom.
5. If this procedure call is not given, the first line is set to one and the last line is set to infinity.

Examples

1. VLIM (10,50)

   Data transmission starts on line 10 and ends on line 50.

2. VLIM (1, TOTAL)

   Data transmission starts on the first line of a page, and ends on the line specified by the value of TOTAL.

EXAMPLES OF LAYOUT PROCEDURES

1.
```
'PROCEDURE' SET'
'BEGIN'
  FORMAT ("3D.2D\);
  'IF' A 'EQ' B↑2 'THEN'
  'BEGIN'
      FORMAT ("ZZZ\);
      TABULATION (5)
  'END';
  VLIM ('IF' A 'EQ' B↑2 'THEN' 5
      'ELSE' 10,50)
'END'
```

   If $A=B^2$, the second format call will override the first, a TAB of 5 will be set and the vertical margins will be (5,50). If $A \neq B$, the first format will be in effect, the TAB will be 1 and the vertical margins will be (10,50).

2.
```
'PROCEDURE' LAYOUT;
'BEGIN'
  FORMAT ("↑,100(ZZD.D,BBD.D'DD),
       /\);
  HLIM(5,60);
  HEND(GOOD,OVER,OVER)
'END';
'PROCEDURE' GOOD; HLIM(5,60);
'PROCEDURE' OVER; HLIM(15,60)
```

Whenever line overflow occurs procedure OVER will change the horizontal margins. When the "/" in the format call is reached, procedure GOOD will restore the original margins.

DATA TRANSMISSION PROCEDURES

These procedures handle the actual transmission of data for input and output.

In calling these procedures it is necessary to specify the I/O device which is to be used for the transmission. File control cards are required to indicate the devices to be used.

Files used by ALGOL are restricted to the numeric file codes 01 to 40 (decimal). 05 is the standard input file and 06 the standard output file. These two files do not require file control cards in order to be used. Unless redefined, file 05 indicates file I*; 06 indicates file P*. Error messages will thus be written on 06.

The point in a program at which the actual I/O procedure is called is when the transmission of data occurs. Layout procedures, if any, and a list procedure, if any, will be called by the internal I/O procedures.

INLIST

To indicate that data is to be transmitted for input; to specify the input device, the setup procedure and the list procedure.

Form

INLIST (a_1,a_2,a_3)

a_1: arithmetic expression

a_2,a_3: procedure identifiers

Rules

1. a_1 is the file number from the file control card which indicates the specific input device to be used.

2. a_2 is the name of the setup procedure containing the layout procedure calls.

3. a_3 is the name of the list procedure which contains the data items to be transmitted.

4. When INLIST is executed, it first calls the layout procedures, then transfers back and forth to the list procedure while the actual input is taking place. See Appendix C for a detailed explanation of INLIST.

Examples

1. INLIST (05, ABC, INPT)

   This statement causes input to take place on I/O device 5 according to the layout procedures in procedure ABC and according to the list procedure INPT.

2.
```
'BEGIN' 'PROCEDURE' START;
   'BEGIN'
      FORMAT("↑,A,D.D'DD/,ZDD,P\);
      VLIM(2,50)
   'END';...
   'PROCEDURE' LIST (OK);
   'BEGIN'
   OK(ALPHA); OK(BOY); OK(COUNT);
   OK(BOOL)
   'END';...
   INLIST (7,START,LIST);...
'END'
```

   This program transmits a symbol into ALPHA, a real number into BOY, an integer into COUNT and a 0 or 1 into BOOL.

INPUT n

To indicate that data is to be transmitted for input; to provide for data input without using layout procedures or a list procedure.

Form

INPUT n (a,string,e_1,e_2,...,e_n)

n: integer

a: arithmetic expression

string: format string

e_1,e_2,...,e_n: variables or subscripted variables

Rules

1. a is the file number from the file control card which indicates the input device to be used.

2. The format string is in the same form as the format call FORMAT (string), i.e., no X's are allowed in the string.

3. $e_1,e_2,...,e_n$ are the actual data items to be transmitted according to the format string given.

4. n may have the value 0,1,2,...,9 and indicates the number of data items.

5. The equivalent of this procedure call in terms of INLIST is as follows:

```
'BEGIN' 'PROCEDURE' LAYOUT; FORMAT (string);
    'PROCEDURE' LIST (ITEM);
    'BEGIN' ITEM (e1); ITEM (e2);...;
          ITEM (en)
    'END';
    INLIST (a,LAYOUT,LIST)
'END'
```

6. When the only layout procedure required is FORMAT and when there are nine or fewer items to be transmitted, this simpler input call may be used instead of INLIST.

Examples

1. INPUT 6 (05, "(ZD.D)↑\,
A[1], A[2], A[3], A[4], A[5], A[6]

   This transmits 6 values according to the repeated format ZD.D.

2. INPUT 2 (07,"P,F\, B[1], B[2])

   This transmits 1 or 0 into B_1 and 'TRUE' or 'FALSE' into B_2.

OUTLIST

To indicate that data is to be transmitted for output; to specify the output device, the setup procedure and the list procedure.

Form

OUTLIST (a_1,a_2,a_3)

a_1: arithmetic expression

a_2,a_2: procedure identifiers

Rules

1. a_1 is the file number from the file control card which indicates the specific output device to be used.

2. a_2 is the name of the setup procedure containing the layout procedure calls.

3. a_3 is the name of the list procedure which contains the data items to be transmitted.

4. When OUTLIST is executed, it first calls the layout procedures then transfers back and forth to the list procedure while the actual output is taking place. See Appendix C for a detailed explanation of OUTLIST.

Examples

1. OUTLIST (10,PAGE,LIST)

   This statement causes output to take place on I/O device 10 according to the layout procedures in procedure PAGE and according to the list procedure LIST.

2.
```
'BEGIN' 'PROCEDURE' SET;
   FORMAT("3D.D,BZZD,2B3S/\);
   ...
   'PROCEDURE' OUT (A);
   'BEGIN'
      A(TOTAL);A(INTEGER);A("ANS\)
   'END';...
   OUTLIST(6,SET,OUT);...
'END'
```

   This program causes the values of the two variables TOTAL and INTEGER and the string ANS to be written out on device 6.

OUTPUT n

To indicate that data is to be transmitted for output; to provide for data output without using layout procedures or a list procedure.

Form

OUTPUT n (a,string,e_1,e_2,...,e_n)

n: integer

a: arithmetic expression

string: format string

e_1,e_2,...,e_n: arithmetic expressions, Boolean expressions or string

Rules

1. a is the file number from the file control card which indicates the output device to be used.

2. The format string is in the same form as the format call FORMAT (string); i.e., no X's are allowed in the string.

3. e_1,e_2,...,e_n are the actual data items to be transmitted according to the format string given.

4. n may have the value 0,1,...,9 and indicates the number of data items.

5. The equivalent of this procedure call in terms of OUTLIST is as follows:

```
'BEGIN' 'PROCEDURE' LAYOUT; FORMAT (string);
    'PROCEDURE' LIST (ITEM);
    'BEGIN' ITEM (e1); ITEM (e2);...;
            ITEM (en)
    'END';
    OUTLIST (a,LAYOUT,LIST)
'END'
```

6. When the only layout procedure required is FORMAT and when there are nine or fewer items to be transmitted, this simpler output call may be used instead of OUTLIST.

Examples

1. OUTPUT 3 (06,"3(2ZD.DD)\,A,B,C)

   This statement will cause 3 values of A, B and C to be transmitted to device 6 according to the format given.

2. OUTPUT 5 (09,"2(D.D'ZZ), 3S,
   2BSS,I\, X↑2-3,Y↑2-3,"TOT\,
   "A1\, COUNT)

   This statement will cause 2 decimal numbers, 2 strings and an internal notation integer to be transmitted.

WTREC

To transfer data from an array in core memory to an external device using logical record handling.

Form

WTREC (FC,A)

FC: arithmetic expression

A: array identifier

Rules

1. FC is the logical file number.

2. A is a one-dimensional array which contains the information to be transferred.

3. The array A must be declared with a lower bound of zero.

4. Before calling WTREC, the element A [0] must have been set to the number of words to be written.

5. The words to be written are stored in A [1] through A $\lceil A[0] \rceil$. The contents of A are not altered by WTREC.

6. The array A will usually be of type 'INTEGER', but in certain cases it may be 'REAL' or 'EXTENDED REAL'.

7. Procedure WTREC controls the transfer via the File and Record Control routines. It also performs any packing of the data as required by the destination codes. The format of the data in memory is discussed for each destination code. (See Appendix D.)

8. Printer - Each word of the array A corresponds to one character for the printer. Each character is stored in the internal character code. These codes are given in the table below. Numbers outside the range 0-63 are treated modulo 64.

   A maximum of 136 characters may be written with each call to WTREC.

   Character codes 15 and 63 have special meaning. Code 15 is ignored completely by the printer--i.e., nothing is printed. (Blank characters have code 16).

   Character code 63 is used to indicate the end of the print line. After printing the actual line, the printer then examines the character following the 63. This character is interpreted as follows:

0-15	the printer is spaced that (0-15) number of lines.
16	the printer is spaced to the top of the next page.
17-63	see PRT201 Printer reference manual, BQ39.

   WTREC ends all lines with the spacing combination 63 and 1, indicating spacing to the next line. However, if the user has given his own spacing combination, his will be encountered first; and the one given by WTREC will be ignored.

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
10	[	#	@	:	>	?	∅	A	B	C
20	D	E	F	G	H	I	&	.	]	(
30	<	\	↑	J	K	L	M	N	O	P
40	Q	R	-	$	*	)	;	'	+	/
50	S	T	U	V	W	X	Y	Z	←	,
60	%	=	"	!						

Line Internal Character Codes

9. Card Punch, Decimal - This is the same as the description for the printer, with the exceptions that the maximum number of characters which may be written is 80 and that the character codes 15 and 63 have no special meaning.

10. Card Punch, Binary - Each element of A corresponds to one column of the punched card. Within each word, bit position 35 goes to row 9, bit position 34 goes to row 8, etc.

11. Paper Tape - Output is always in the binary mode. Each element of array A corresponds to one row on the tape. For further information see PTS200 Perforated Tape Subsystem reference manual, BP44.

12. Magnetic Tape - Output occurs word for word. For these destination codes, the array A may be of type 'REAL' or 'EXTENDED REAL'. Note that, for 'EXTENDED REAL', A [0] must contain twice the number of elements to be transferred.

WTCHAR

To insert one character in the current logical output record.

Form

WTCHAR (FC,A)

FC,A: arithmetic expression

Rules

1. FC is the logical file number.

2. The actual value of A is the character to be inserted.

3. The format of the character in A is determined by the final destination code. The format is the same as discussed under Rule 7 of WTREC. Of course, with WTCHAR, A represents only one value.

4. The end of the current logical record is indicated by calling WTREC with a negative value of A. When the final destination code refers to the printer and the value of A is negative, the absolute value of A is the number of lines to be spaced.

RDREC

To transfer data from an external device to an array in memory using logical record handling.

Form

RDREC (FC,A)

FC: arithmetic expression
A: array identifier

Rules

1. FC is the logical file number.

2. A is a one-dimensional array which will contain the information to be read.

3. The array must be declared with a lower bound of zero.

4. RDREC will set A [0] to the number of words read on each call.

5. A [0] will be set to -1 if RDREC encounters an end-of-file condition.

6. The information read will be stored in A [1] through A[⌈A 0⌉].

7. The array A will usually be of type 'INTEGER', but in certain cases it may be 'REAL' or 'EXTENDED REAL'.

8. Procedure RDREC performs any packing of the data as required by the destination code. The format of the data in memory is discussed for each destination code. (See Appendix D.)

9. Card Reader - In reading from cards, column 1 goes into A [1], column 2 into A [2], etc. With BCD cards, the number value read ranges from 0 through 63 according to the table listed under WTREC. For binary cards the number ranges from 0 through 4095.

10. Paper Tape - Input is always in the binary mode. Each element of array A corresponds to one row on the tape. For further information see PTS200 Perforated Tape Subsystem reference manual, BP44.

11. Magnetic Tape - Input occurs word for word. For these destination codes, the array A may be of type 'REAL' or 'EXTENDED REAL'. Note that, for 'EXTENDED REAL', A [0] must contain twice the number of elements to be transferred.

Note also that, when reading, the type of the array A must be the same as that from which the information was written, as no type conversion is done by RDREC or WTREC.

RDCHAR

To read one character from the current input logical record.

Form

A ← RDCHAR (FC)

FC: arithmetic expression

Rules

1. FC is the logical file number.

2. The value returned will be the character read. The format of the character is determined by the final destination code. The format is the same as discussed under Rule 8 of RDREC.

3. When an end-of-file condition is encountered, RDCHAR returns a value of -1.

TRANSMIT

To transfer data between core memory and an external device (magnetic tape, disk, or drum) using physical record handling.

Form

I ← TRANSMIT (FC, INDEX, MODE, LIST)

FC, INDEX, MODE: arithmetic expressions

LIST: procedure identifier

Rules

1. FC is the logical file number.

2. INDEX is the number of the first word within the file from which transferring is to begin. (INDEX $\geq$ 0.)

3. MODE indicates whether this is input or output and also indicates the type of the variables as they appear on the external device.

4. LIST is a procedure which specifies the variables to be transmitted.

5. File control block for files used with TRANSMIT must indicate physical record handling--i.e., no buffers are used.

6. Files for disk and drum must be of type random.

7. The parameter MODE indicates the type of the variables as they exist or are to exist on the external device. The possible values of MODE are given in the following table.

Type	Read	Write
'REAL'	0	1
'INTEGER'	2	3
'EXTENDED REAL'	4	5

8. The mode is entirely independent of the type of the variables as they appear in memory, because the necessary conversion occurs in the list procedure.

9. The beginning of the information to be referenced on the external device is indicated by the value of the parameter INDEX. This is simply a sequential counter of the words as they are written on or read from the external device. The following two rules define the value of INDEX.

   a. For the first item (i.e., variable) to be referenced INDEX = 0.

   b. INDEX is incremented by 1 for each single-word variable ('INTEGER' and 'REAL') and by 2 for each double-word variable ('EXTENDED REAL').

10. The function value returned by TRANSMIT is the next unused INDEX. This can be used in a loop of the following type:

```
INDEX: = TRANSMIT (17, INDEX, 1, LIST)
```

11. Physical record handling is used. This means that data is transmitted with each call to TRANSMIT. Therefore, buffers are not found for files to be used with TRANSMIT. Temporary buffer space is reserved in the stack with each call, and this space is released upon return. At the time that the first reference to a file is made, it is opened by calling .UOPEN. Upon termination of the program the file is closed by .AEXIT.

The methods for positioning and the record format are slightly different for magnetic tape than for disk or drum.

Magnetic Tape

Each call to TRANSMIT results in transmitting at least one physical record. Each record begins with a control word. Data follows the control word. The contents of the control word are:

Bits 0-23 INDEX for the next record
Bits 24-35 MODE for this record

The maximum length of any one physical record is 320 words, not counting the control word. For calls to TRANSMIT which indicate more than 320 words, the required number of 320-word records will be transmitted.

Before reading or writing, TRANSMIT checks the INDEX against the previous value and then positions the tape accordingly.

The following restrictions must be observed:

1. Reading beyond the limits of the latest write operation is not possible.
2. Writing beyond the limits of the latest write operation is not possible. Writing occurs only with the next available INDEX--i.e., no holes are allowed.
3. Reading and rewriting always start with an INDEX which is the same as the INDEX from an earlier writer.
4. For the initial writing of data on tape, the INDEX must be zero.

Violation of any of the above rules causes TRANSMIT to print an error message and to terminate the execution.

Drum and Disk

The restrictions for tape handling are removed for disk and drum. The parameter INDEX is used to calculate a starting address within the links allocated for the disk or drum. Thereafter, transmission occurs from this point.

There is no special control word at the beginning of each record. This means that the parameter MODE must have the same type representation when reading a record as it had when the record was written. For example, a file may not be written with the MODE as 'REAL' and then later read as 'INTEGER'.

INPUT/OUTPUT CONTROL PROCEDURE

The following input/output control procedure accesses system parameters and allow some control over the positioning of the I/O devices.

SYSPARAM

To gain access to certain system parameters so that they may be modified.

Form

SYSPARAM (a_1,a_2,a_3)

a_1,a_2: arithmetic expression

a_3: integer variable

Rules

1. The system parameters which may be changed or read out are:

   a. The character, line and page pointers (p, p' and p") respectively. p and p' have an initial value of zero. E.g., when p=2, the third character is the next character to be accessed.

   b. The "standard format" constant determining the number of spaces between items (k).

   c. The physical end of line (P) and the physical end of page (P') which are characteristic of the I/O device.

2. a_1 is the file number from the file control card specifying the I/O device concerned.

3. a_2 may have a value of 1,2,...,11.

   a. If the value of a_2 is 1,3,5,7,9 or 11, the value of the system parameter in question is assigned to variable a_3.

   b. If the value of a_2 is 2,4,6,8 or 10, the value of a_3 becomes the new value of the system parameter.

   c. The action is as follows:

if a_2 = 1, $a_3 \leftarrow$ p	if a_2 = 2, p $\leftarrow a_3$
if a_2 = 3, $a_3 \leftarrow$ p'	if a_2 = 4, p' $\leftarrow a_3$
if a_2 = 5, $a_3 \leftarrow$ P	if a_2 = 6, P $\leftarrow a_3$
if a_2 = 7, $a_3 \leftarrow$ P'	if a_2 = 8, P' $\leftarrow a_3$
if a_2 = 9, $a_3 \leftarrow$ k	if a_2 = 10, k $\leftarrow a_3$
if a_2 = 11, $a_3 \leftarrow$ p"	

4. p and p' represent actual positions on the I/O device which are to be changed when a_2 = 2 and a_2 = 4.

   a. If a_2 = 2, p is tested to see if $p<a_3$. If it is, blanks are inserted until p = a_3. If $p \geq a_3$, a skip to the next line is performed, p is set equal to 0, and blanks are inserted until p = a_3.

b. If $a_2 = 4$, p' is tested to see if $p' < a_3$. If it is, lines are advanced until $p' = a_3$. If $p' \geq a_3$, a skip is made to a new page, p' is set equal to 0, and lines are advanced until $p' = a_3$.

5. $a_2 = 6$ and $a_2 = 8$ change the physical limits of the I/O device (P and P') where this is possible (i.e., magnetic tape block length may be changed and unit record devices may have physical limits reduced, but not extended beyond the standard limits). If the limits cannot be changed and these actions are specified, the statement acts like a dummy statement.

6. a_2 may also have a value of 21 or 22

   a. If the value of a_2 is 21, the file denoted by a_1 is defined as an input file.

   b. If the value of a_2 is 22, the file denoted by a_1 is defined as an output file.

   c. If the value of a_2 is 21 or 22, the value of a_3 is not significant.

7. The condition which requires $a_2 = 21$ or 22 only arises when the intended first action on a particular file uses a primitive procedure or procedure SYSPARAM. If this is the case, the system does not know the nature of the file and thus a call to SYSPARAM with $a_2 = 21$ or 22 would serve to define the file; e.g., if the first action with respect to file 6 is to read out the value of p by a call to SYSPARAM such as

```
SYSPARAM(06, 1, CHAR)
```

This call would have to be preceded by a call to SYSPARAM defining 06 as an output file as follows:

```
SYSPARAM(06, 22, 0)
```

8. a_2 may also have a value of 23. If a_2 is 23 then the file will be closed with the options indicated by a_3. The value of a_3 (which is treated modulo 4) indicates:

   0 Rewind but do not lock the file
   1 Do not rewind and do not lock the file
   2 Lock and release the file.
   3 Rewinding and/or locking is not appropriate for this device.

9. a_2 may also have a value of 24. If a_2 is 24 then a may have a value of 0 through 7. This function acts like the DSTCOD on the $ FFILE control card. The appropriate values in the file parameter list (maximum characters per line, number of characters per word, media code, ASCII/BCD flag, etc.) will be set according to the value of a_3.

Value of a	Action
0	Binary (MTAPE, DISC, DRUM)
1	Binary card (BCRDR, BCPNCH)
2	BCD card (DCRDR, DCPNCH)
3	BCD printer (PRNTR)
4	ASCII printer
5	Paper tape single (PTMODS)
6	Paper tape double (PTMODD)
7	Paper tape edit (PTMODE)

Examples

1. SYSPARAM (8, 3, LINENO)

   On device 8 the value of the line pointer is assigned to variable LINENO.

2. SYSPARAM (5, 10, 3)

   For device 5 the value of k is changed to 3; i.e., 3 or more blanks must follow a number in standard format.

PRIMITIVE PROCEDURES

The following primitive procedures are available for use by the programmer but are not intended to be general purpose routines.

INSYMBOL

To associate specific ALGOL symbols with specific integers; to read in a basic symbol from an external device as an integer.

Form

INSYMBOL (e, s, v)

e: arithmetic expression
(called by value)

s: string

v: integer variable

Rules

1. The basic symbols contained in the string "s" are given integer values.

2. The symbols are assigned from left to right to the positive integers 1,2,3, etc.

3. This procedure acts as follows:

   a. It reads in the next symbol from the input device.

   b. If it is a basic symbol which appears in the string "s", the variable v will be assigned the integer value associated with this symbol.

   c. If it is a basic symbol which does not appear in the string "s", v will receive a value of 0.

d. If the input symbol is not an ALGOL basic symbol, v will receive a value of minus one.

e. If there is no more data on the input device, v will receive a value of minus two.

f. If the string "s" is null, i.e., INSYMBOL (e,"\,v), v will receive the standard system value for the basic symbol. (See Appendix E.)

LENGTH

To calculate the length of a given string.

Form

LENGTH (s)

s: string

Rules

1. The result of this procedure is an integer.

2. It is equal to the number of basic symbols in the string "s" not including the outermost pair of string quotes.

NAME

To permit the saving or "remembering" of labels and procedure identifiers.

Form

NAME (v_1,v_2,a,p)

v_1,v_2: integer variable

a: statement label

p: procedure identifier

Rules

1. If v_1 has a value of 1, the integer associated with a is assigned to v_2.

2. If v_1 has a value of 3, the integer associated with p is assigned to v_2.

3. If v_1 has a value of 2, control will be transferred to the label whose value is the same as that of v_2. (Note: v_2 must have been assigned the value of a label by a previous NAME statement.) If $v_2 = 0$ the program will be terminated.

4. If v_1 has a value of 4, control will be transferred to the procedure whose identifier has the same value as v_2. (Note: v_2 must have been assigned the value of a procedure identifier by a previous NAME statement.) If $v_2 = 0$ the procedure will be a dummy procedure.

5. The association of specific integers with labels and procedure identifiers holds only in the block in which the labels or identifiers are declared, i.e., the rules of scope for ALGOL block structure are obeyed.

OUTSYMBOL

To associate ALGOL basic symbols with specific integers; to write out a basic symbol on an external device from an internally stored integer.

Form

OUTSYMBOL (e_1, s, e_2)

e_1,e_2: arithmetic expression
(called by value)

s: string

Rules

1. The basic symbols in the string "s" are given integer values.

2. The positive integers 1,2,3, etc. are assigned to the symbols from left to right; leftmost = 1, next = 2, etc.

3. This procedure acts as follows:

   a. It evaluates e_2 and determines the integer which is closest to this value.

   b. If the value has an equivalent in string "s", the basic symbol corresponding to this value will be written on the output device.

   c. If the value has no equivalent in string "s" by being outside the bounds of the string, or if it is not a basic symbol, the ∅ will be written on the output device.

   d. If the string "s" is null, i.e., OUTSYMBOL (e_1,"\,e_2), the standard system values are used to determine the basic symbol which will be written on the output device. (See Appendix E.)

STRING ELEMENT

To enable the scanning of a given string (actual or formal) in a machine independent manner.

Form

STRING ELEMENT (s_1,v_1,s_2,v_2)

s_1,s_2: string

v_1,v_2: variable

Rules

1. Variable v_1 determines which symbol of s is referenced, i.e., if v_1 = 1 it is the leftmost symbol; if v_1 = 2, the next, etc.

2. Once the symbol is chosen, its associated integer is assigned to variable v2. (Note: the associated integer is determined by encloding string s_2 as was done with the string in the procedure INSYMBOL.)

TYPE

To determine the type of a number which is to be written out in standard format.

Form

TYPE (v_1,v_2)

v_1: variable

v_2: variable or string

Rules

1. If v_2 is a string, v_1 is set equal to 4.

2. If v_2 is a variable, v_1 is assigned a different value depending on the type of v_2 as follows:

   a. If v_2 is 'INTEGER', $v_1 \leftarrow 1$.

   b. If v_2 is 'REAL', $v_1 \leftarrow 2$.

   c. If v_2 is 'BOOLEAN', $v_1 \leftarrow 3$.

   d. If v_2 is 'EXTENDED REAL', $v_1 \leftarrow 8$.

LIST PROCEDURE

This procedure is written by the programmer to be used with the I/O procedures provided by ALGOL. It provides a list of the data items to be transmitted. The identifier for this procedure is not reserved by ALGOL; thus, any valid identifier may be chosen.

List Procedure

To list a sequence of quantities to be transmitted for input or output; this list is used in conjunction with the format items of a FORMAT call.

Form

'PROCEDURE' name (ident); s

name: procedure identifier

ident: identifier

s: simple statement or block

Rules

1. The formal parameter "ident" appears in the body of the list procedure as a procedure identifier.

2. Each item to be transmitted for input or output appears in the procedure body as the parameter for procedure ident.

   Example:

   'PROCEDURE' A(X); 'BEGIN' X(M); X(N)' X(P) 'END'

   M, N and P are transmitted.

3. When the list procedure is called by a data transmission procedure (INLIST or OUTLIST), an internal system procedure (INITEM or OUTITEM) will be the actual parameter corresponding to the formal parameter "ident," and thus will be substituted for "ident" in the list procedure body.

4. Execution of the list procedure causes the internal system procedure (INITEM or OUTITEM) to be executed. INITEM or OUTITEM has as its parameter the item to be transmitted.

5. This parameter may be an arithmetic expression, Boolean expression or a string for output. However, the parameter may be only a variable or subscripted variable for input.

6. The item is called by name by the internal system procedure and its value is transmitted for input or output.

7. The sequence of statements in the list procedure body determines the sequence in which the items are transmitted for input or output.

8. All ALGOL statements are permissible in a list procedure including a call to one or more of the layout procedures.

Examples

1.
```
'PROCEDURE' LIST (NAME);
'BEGIN' NAME(X); NAME (Y↑3*Z);
NAME ("TOTAL\) 'END'
```

The identifier NAME is replaced by a system procedure name when the list procedure is called. X,Y↑3*Z and "TOTAL\ are parameters to this system procedure and their values will be transmitted.

2.
```
'PROCEDURE' MANY (ITEM);
'FOR' I←1 'STEP' 1 'UNTIL'
10 'DO' 'BEGIN' ITEM (A[I]);
ITEM (B[I]) 'END'
```

The items to be transmitted are $A_1, B_1, A_2, B_2, \ldots, A_{10}, B_{10}$.

APPENDIX A.

RESERVED IDENTIFIERS

The following list enumerates reserved identifiers. These identify functions and procedures which are available without explicit declarations. These functions and procedures are assumed to be declared in a block external to the program. However, a programmer may redeclare a reserved identifier, in which case the reserved meaning is superseded.

The reserved identifiers are as follows:

ABS
ARCTAN
BAD DATA
CLOCK
COS
DATE
DUMP
ENTIER
EQUIV
ERRNO
EXP
FORMAT
FORMAT 0
FORMAT 1
FORMAT 2
FORMAT 3
FORMAT 4
FORMAT 5
FORMAT 6
FORMAT 7
FORMAT 8
FORMAT 9
HEND
HLIM
INLIST
INPUT 0
INPUT 1
INPUT 2
INPUT 3
INPUT 4
INPUT 5
INPUT 6
INPUT 7
INPUT 8
INPUT 9
INSYMBOL
LENGTH
LN
NAME
NO DATA
OUTLIST
OUTPUT 0
OUTPUT 1
OUTPUT 2
OUTPUT 3
OUTPUT 4
OUTPUT 5
OUTPUT 6
OUTPUT 7
OUTPUT 8
OUTPUT 9
OUTSYMBOL
PACK
RDCHAR
RDREC
SENSE
SIGN
SIN
SPACE
SQRT
STRINGELEMENT
SYSPARAM
TABULATION
TIME
TRANSMIT
TYPE
UNPACK
VEND
VLIM
WTCHAR
WTREC

APPENDIX B.

MATHEMATICAL AND MISCELLANEOUS FUNCTIONS

MATHEMATICAL FUNCTIONS

Form	Description
ABS (e)	absolute value of the expression e
ARCTAN (e)	principal value of the arctangent of e
COS (e)	cosine of e
ENTIER (e)	the integral part of e
EXP (e)	exponential function of e
LN (e)	natural logarithm of e
SIGN (e)	sign of e (+1 if e>0, 0 if e = 0, -1 if e<0)
SIN (e)	sine of e
SQRT (e)	square root of e

These functions are available without explicit declarations. They are assumed to be declared in a block external to the program. However, a programmer may redeclare a mathematical function identifier, in which case the standard meaning is superseded.

These functions accept parameters of types 'REAL', 'EXTENDED REAL' and 'INTEGER'. They all yield values of type 'EXTENDED REAL', except for ENTIER(e) and SIGN(e) which yield values of type 'INTEGER'.

The parameters of these functions are treated as 'VALUE' parameters.

MISCELLANEOUS FUNCTIONS

DATE is an integer procedure which returns the current date expressed as an integer. For example, August 1, 1968, would be returned as 80,168.

SPACE is an integer procedure which returns the number of words of memory available in the stack.

CLOCK is an extended-real procedure which returns the time of day expressed in hours past midnight.

TIME is an extended-real procedure which returns the amount of processor time expended, expressed in seconds.

SENSE (e) is a Boolean procedure which returns the setting of the program switch word for the bit indicated by the parameter e. The parameter is an arithmetic expression. It has a value from 1 to 6 corresponding to the options ON1 through ON6 on the $ EXECUTE card.

ERRNO (e) performs one of two functions:

1. If the value of the parameter e is positive, a message of the form < < < ERROR NUMBER XX > > > is printed, where XX is the value of the parameter.

2. If the value is negative, it replaces the maximum error count (which is the maximum number of error messages which may be printed before the execution is aborted). The maximum error count is currently five.

DUMP is a procedure which gives a trace of the procedural and nonprocedural stacks. The ON6 option on the $ EXECUTE control card must also be used. If the ON6 option has not been used, then the call is ignored. This allows the user to call DUMP in the source program, but to determine the need for the trace at execution time.

PACK

To pack bits together into a word.

Form

PACK (S,I,L);

S: string

I: identifier

L: list procedure

Rules

1. The routine packs bits coming from the list procedure into I according to the string S.

2. I is the identifier of the resulting packed word.

3. L is a procedure which lists the values which are to be packed together.

4. The string consists of a set of integers separated by commas.

   "n1,n2,n3,...\

5. The rightmost n1 bits of the first value received from the list procedure will be stored into the leftmost n1 bits of the variable I.

6. The rightmost n2 bits of the second value received from the list procedure will be stored into the next n2 bits (after n1) of the variable I.

7. An integer in the string may have a trailing Z. This causes zeros to be placed in that field in the output variable. In addition, the list procedure is not invoked for that field.

Examples

1.
```
'PROCEDURE' LIST(P); 'BEGIN'
    P(6);P(5);P(4);P(3);P(2);P(1); 'END';
'INTEGER' I;
PACK ("6,6,6,6,6,6\,I, LIST);
```

This would result in: I-060504030201

2.
```
'INTEGER' I,J,K;
'PROCEDURE' LIST(P); 'BEGIN'
    P(I);P(J); 'END'
I:=70;J:=42;
PACK("9,21\,K, LIST);
```

This would result in: K+106000005200 (remembering that 70 decimal is 106 octal, and 42 decimal is 52 octal).

3.
```
'INTEGER' I,J,K;
'PROCEDURE' LIST(P); 'BEGIN'
        P(I);P(J); 'END';
I:=63;J:=3;
PACK("3Z,6,3\,K,LIST);
```

This would result in: K=077300000000 (again remembering that 63 decimal is 77 octal).

UNPACK

To unpack parts (bits) of a word into separate words.

Form

UNPACK(S,I,L);

S: string

I: identifier

L: list procedure

Rules

1. The routine unpacks bits from I into variables given by the list procedure according to the string S.

2. I is the identifier of the word to be unpacked.

3. L is the procedure which lists the variables which are to receive the unpacked bits.

4. The string consists of a set of integers separated by commas.

```
"n1,n2,n3,...\
```

5. The leftmost n1 bits of the variable I are stored into the rightmost n1 bits of the first variable given in the list procedure.

6. Then the next n2 bits (after n1) of the variable I are stored into the rightmost n2 bits of the second variable given in the list procedure.

Example

```
'REAL' X; 'INTEGER' I,J;
'PROCEDURE' LIST(P); 'BEGIN'
    P(I);P(J); 'END';
    X:=1.5;
UNPACK("9,27\,X,LIST);
```

This would result in:

```
I=000000000002
J=000600000000
```

APPENDIX C.

DETAILED EXPLANATION OF INLIST AND OUTLIST

INLIST

Assume:

1. INLIST has been called.
2. Lines 1,2,...,p' of the current page have been read.
3. Characters 1,2,...,p of the current line (line p' + 1) have been read.
4. At the beginning of the program p = p' = 0.
5. Symbols P and P' denote line size and page size respectively.
6. There are eight hidden variables H1, H2,...H8 which correspond to the eight layout procedures as follows:

   H1 - FORMAT
   H2 - HLIM
   H3 - VLIM
   H4 - HEND
   H5 - VEND
   H6 - TABULATION
   H7 - NO DATA
   H8 - BAD DATA

7. The left margin of HLIM is L.
   The right margin of HLIM is R.
   The top margin of VLIM is L'.
   The bottom margin of VLIM is R'.

STEP 1. (Initialization)

The hidden variables are set to standard values:

H1 is set to the "standard" format.

H2 is set so that L = 1, R = ∞.

H3 is set so that L' = 1, R' = ∞.

H4 is set so that the three parameters are all effectively equal to the dummy procedure defined as follows: 'PROCEDURE' DUMMY;;.

H5 is set so that the three parameters are all effectively equal to the dummy procedure, DUMMY.

H6 is set so that TAB = 1.

H7 is set to terminate the program in case the data ends.

H8 is set to terminate the program if an unacceptable character is received for format translation.

STEP 2. (Layout)

The layout procedure is called; this may change some of the variables H1, H2, H3, H4, H5, H6, H7, H8. Set T to 'FALSE'. (T is a Boolean variable used to control the sequencing of data with respect to title formats; T = 'TRUE' means a value has been requested of the procedure which has not yet been input.)

STEP 3. (Communication with List Procedure)

The next format item of the format string is examined. (Note: after the format string is exhausted, "standard" format is used from then on until the end of the procedure. In particular, if the format string is " , standard format is used throughout.) Now if the next format item is a title format, that is, requires no data item, we proceed directly to Step 4. If T = 'TRUE', proceed to Step 4. Otherwise, the list procedure is activated. This is done the first time by calling the list procedure, using as actual parameter a procedure named IN ITEM.

This is done on all subsequent times by merely returning from the procedure IN ITEM which will cause the list procedure to be continued from the latest IN ITEM call. (Note: the identifier IN ITEM has scope local to IN LIST so a programmer may not call this procedure directly.) After the list procedure has been activated in this way, it will either terminate or will call the procedure IN ITEM. In the former case, the input process is completed; in the latter case, T is set to 'TRUE'. Then any assignments to hidden variables that the list procedure may have invoked will cause adjustment to the variables H1, H2, H3, H4, H5, H6, H7, H8, (which are local to IN ITEM). We then continue at Step 4.

STEP 4. (Alignment Marks)

If the next format item includes alignment marks at its left, remove them from the format and execute process A (a subroutine below) for each "/", process B for each "↑", and process C for each "J".

STEP 5. (Get within Margins)

Execute process G to ensure proper page and line alignment.

STEP 6. (Formatting for Input)

Take the next item from the format string.

NOTES:

In unusual cases, the list procedure or an overflow procedure may have called the descriptive procedure FORMAT thereby changing the format string. In such cases, the new format string is examined from the beginning; and it is conceivable that the format items examined in Steps 3, 4, and 6 might be three different formats. But at this point the current format item is effectively removed from the format string and copied elsewhere so that the format string itself, possibly changed by further calls of FORMAT, will not be interrogated until the next occurrence of Step 3.

Alignment marks at the left of the format item are ignored. If the format item is not composed only of alignment marks and insertions, the value of T is examined. If T = 'FALSE', undefined action takes place (the programmer has substituted a nontitle format for a title format in an overflow procedure, and this is not allowed). Otherwise, T is set to 'FALSE'. If the format item is "A" or "N", set s = 1 and go to Step 7; otherwise, the number of characters, s, needed to input the format item for the present medium is determined, and it is assumed that the same number of character positions will be used in the input medium for this item.

STEP 7. (Check for Overflow)

If the present item uses "N" format, the character positions p + 1, p + 2,... are examined until either a proper termination of the number has been found, or position min (R,P) has been reached with no sign, digit, decimal point, or "'" encountered. In the former case, set s to the number of character positions occupied by the number, including preceding and embedded blanks and the termination character, and then go to Step 9; in the latter case, go to Step 8 with p = min (R,P).

If the present item uses "A" format, the character position p + 1 is examined, if it contains "γ", set p = min (R,P) and go to Step 8, otherwise input characters starting from position p + 1 until a basic symbol has been input. Set s to the number of characters denoting the basic symbol and go to Step 9. Finally, if neither "N" nor "A" format is used, go to Step 8 or Step 9 according as p + s > min (R,P) or not.

STEP 8. (Processing of Overflow)

Perform process H (p + s). Then proceed as follows:

"N" format: Input characters until either finding a number followed by a proper termination (go to Step 9) or until reaching position min (R,P). In the latter case, a partial number may have been examined; repeat Step 8 until a number properly terminated has been input. In the former case, set s to the number of positions occupied by that portion of the number lying to the right of p, including embedded blanks and the termination character, then go to Step 9.

"A" format: Input characters as with "N" format until a basic symbol has been input. (This basic symbol necessarily takes several character positions on the medium.)

Other: If p + s < R and p + s ≤ P, go to Step 9; otherwise input k = min (R,P) - p characters, set p = min (R,P), decrease s by k, and repeat this step.

STEP 9. (Finish the Item)

If neither "A" nor "N" format is being used, input s characters. Determine the value of the item that was input here, or Steps 7 and 8 in case of "A" or "N" format, using the rules of format. Assign this value to the actual parameter of IN ITEM unless a title format was specified. Increase p by s.

Any alignment marks at the right of the format item now cause activation of process A for each "/", process B for each "↑", and process C for each "J". Return to Step 3.

PROCESS A. ("/" Operation)

Check page alignment with process F, then execute process D and call procedure p1 of HEND.

PROCESS B. ("↑" Operation)

If p > 0, execute process D and call procedure p1 of HEND. Then execute process E and call procedure p1 of VEND.

PROCESS C. ("J" Operation)

Check page and line alignment with process G. Then let k = ((p - L + 1)% TAB + 1) x TAB + L - 1 (the next "tab" setting for p), where TAB is the "tab" spacing for this channel. If k > min (R,P), perform process H(k); otherwise skip over character positions until p = k.

PROCESS D. (New Line)

Skip the input medium to the next "line", set p = 0, and set p' = p' + 1.

PROCESS E. (New Page)

Skip the input medium to the next "page", and set p = 0.

PROCESS F. (Page Alignment)

If p' + 1 < L', execute process D until p' = L' - 1.
If p' + 1 > R', execute process E, call procedure p2 of VEND and repeat process F.
If p' + 1 > P', execute process D, call procedure p3 of VEND and repeat process F.
This process must terminate because $1 \leq L' \leq R'$ and $1 \leq L' \leq P'$. If a programmer chooses a value of L' > P', L' is set equal to 1.

PROCESS G. (Page and Line Alignment)

Execute process F. Then,
If p + 1 < L, skip over character positions until p + 1 = L.
If p + 1 > R or p + 1 > P, perform process H (p + 1).
This process must terminate because $1 \leq L \leq R$ and $1 \leq L \leq P$. If a programmer chooses a value of L > P, L is set equal to 1.

PROCESS H(k). (Line Overflow)

Perform process D. If k > R, call procedure p2 of HEND; otherwise call p3. Then perform process G to ensure page and line alignment. Note: upon return from any of the overflow procedures, and assignments to hidden variables that have been made by calls on descriptive procedures will cause adjustment to the corresponding variables H1, H2, H3, H4, H5, H6, H7, H8.

EXAMPLE:

Notice that the programmer has the ability to determine the presence of absence of data on a card when using standard format, because of the way overflow is defined. The following program, for example, will count the number n of data items on a single input card and will read them into A[1], A[2],...,A[n]. (Assume unit 5 is a card reader.)

```
'PROCEDURE' LAY; HEND (EXIT, EXIT, EXIT);
'PROCEDURE' LIST (ITEM); ITEM (A[N+1]);
'PROCEDURE' EXIT; 'GOTO' L2;
N←0; LI: INLIST (5, LAY, LIST);
N←N + 1; 'GOTO' LI;
L2:;
```

OUTLIST

Assume:

1. OUTLIST has been called.
2. Lines 1,2,...,p' of the current page have been completed.
3. Characters 1,2,...,p of the current line (line p' + 1) have been completed.
4. At the beginning of the program p = p' = 0.
5. Symbols P and P' denote the line size and page size respectively.
6. There are eight hidden variables H1,H2,...,H8 which correspond to the eight layout procedures as follows:

H1 - FORMAT
H2 - HLIM
H3 - VLIM
H4 - HEND
H5 - VEND
H6 - TABULATION
H7 - NO DATA
H8 - BAD DATA

7. The left margin of HLIM is L.
The right margin of HLIM is R.
The top margin of VLIM is L'.
The bottom margin of VLIM is R'.

STEP 1. (Initialization)

The hidden variables are set to the following standard values:

H1 is set to the "standard" format.

H2 is set so that L = 1, R = ∞.

H3 is set so that L' = 1, R' = ∞.

H4 is set so that the three parameters are all effectively equal to the dummy procedure defined as follows: 'PROCEDURE' DUMMY;;.

H5 is set so that the three parameters are all effectively equal to the dummy procedure, DUMMY.

H6 is set so that TAB = 1.

STEP 2. (Set-Up)

The set-up procedure is called; this may change some of the variables H1, H2, H3, H4, H5, H6. Set T to 'FALSE'. (T is a Boolean variable used to control the sequencing of data with respect to title formats; T = 'TRUE' means a value has been transmitted to the procedure which has not yet been output.)

STEP 3. (Communication with List Procedure)

The next format item of the format string is examined. (Note: after the format string is exhausted, "standard" format is used from then on until the end of the procedure. In particular, if the format string is " , standard format is used throughout.) Now if the next format item is a title format, that is, requires no data item, we proceed directly to Step 4. If T = 'TRUE' proceed to Step 4. Otherwise, the list procedure is activated; this is done the first time by calling the list procedure, using as actual parameter a procedure named OUT ITEM; this is done on subsequent times by merely returning from the procedure OUT ITEM, which will cause the list procedure to be continued from the latest OUT ITEM call.

(Note. The identifier OUT ITEM has scope local to OUT LIST, so a programmer may not call this procedure directly.) After the list procedure has been activated in this way, it will either terminate or will call the procedure OUT ITEM. In the former case, the output process is completed; in the latter case, T is set to 'TRUE' and any assignments to hidden variables that the list procedure may have invoked will cause adjustment to the variables H1, H2, H3, H4, H5, H6 (which are local to OUT ITEM) and we then continue to Step 4.

STEP 4. (Alignment Marks)

If the next format item includes alignment marks at its left remove them from the format and execute process A (a subroutine below) for each "/", process B for each "↑", and process C for each "J".

STEP 5. (Get Within Margins)

Execute process G to ensure proper page and line alignment.

STEP 6. (Formatting the Output)

Take the next item from the format string.

NOTES:

In unusual cases, the list procedure or an overflow procedure may have called the descriptive procedure FORMAT, thereby changing the format string. In such cases, the new format string is examined from the beginning, and it is conceivable that the format items examined in Steps 3, 4, 6 might be three different formats. But at this point the current format item is effectively removed from the format string and copied elsewhere, so that the format string itself, possibly changed by further calls of FORMAT, will not be interrogated until the next occurrence of Step 3.

Alignment marks at the left of the format item are ignored. If the format item is not composed only of alignment marks and insertions, the value of T is examined. If T = 'FALSE', undefined action takes place (the programmer has substituted a nontitle format for a title format in an overflow procedure, and this is not allowed). Otherwise, the output item is evaluated and T is set to 'FALSE'. Now the rules of format are applied, and the characters $X_1 X_2 \ldots X_s$ which represent the formatted output on the external medium are determined. (Note that the number of characters, s, may depend on the value being output, using "A" or "S" format, as well as on the output medium.)

STEP 7. (Check for Overflow)

If $p + s \leq R$ and $p + s \leq P$, where s is the size of the item as determined in Step 6, the item will fit on this line, so go on to Step 3. Otherwise, if the present item uses "A" format, output a special symbol "γ" which is recognizably not a basic symbol; this is done to ensure the input will be inverse to output. Go to Step 8.

STEP 8. (Processing of Overflow)

Perform process H (p + s). Then, if $p + s \leq R$ and $p + s \leq P$, go to Step 9; otherwise let $k = \min (R,P) - p$. Output $X_1 X_2 \ldots \overline{X}_K$, set $p = \min (R,P)$ and then let $X_1 X_2 \ldots X_{S-K} = X_{K+1} + X_{K+2} \ldots X_S$. Decrease s by k and repeat Step 8.

STEP 9. (Finish the Item)

Output $X_1X_2...X_S$, and increase p by s. Any alignment marks at the right of the format item now cause activation of process A for each "/", process B for each "↑", and process C for each "J". Return to Step 8.

PROCESS A. ("/" Operation)

Check page alignment with process F, then execute process D and call procedure p1 of HEND.

PROCESS B. ("↑" Operation)

If $p > 0$, execute process D and call procedure p1 of HEND. Then execute process E and call procedure p1 of VEND.

PROCESS C. ("J" Operation)

Check page and line alignment with process G. Then let k = ((p-L+1) % TAB + 1) x TAB + L - 1 (the next "tab" setting for p), where TAB is the "tab" spacing for this channel. If $k \geq \min$ (R,P), perform process H(k); otherwise effectively insert blanks until p = k.

PROCESS D. (New Line)

Skip the output medium to the next "line", set p = 0, and set p' = p' + 1.

PROCESS E. (New Page)

Skip the output medium to the next "page", and set p' = 0.

PROCESS F. (Page Alignment)

If p' + 1 < L' execute process D until p' = L' - 1. If p' + 1 > R', execute process E, call procedure p2 of VEND, and repeat process F.

If p' + 1 > P', execute process E, call procedure p3 of VEND and repeat process F.

This process must terminate because $1 \leq L' \leq R'$ and $1 \leq L \leq P'$. If a programmer chooses a value of L' > P', L' is set equal to 1.

PROCESS G. (Page and Line Alignment)

Execute process F. Then if $p + 1 < L$, effectively output blank spaces until $p + 1 = L$.

If $p + 1 > R$ or $p + 1 > P$, perform process H $(p + 1)$.

This process must terminate because $1 \leq L \leq R$ and $1 \leq L \leq P$. If a programmer chooses a value of $L > P$, L is set equal to 1.

PROCESS H(k). (Line Overflow)

Perform process D. If $k > R$, call procedure p2 of HEND; otherwise, call procedure p3 of HEND. Then perform process G to ensure page and line alignment. Note: upon return from any of the overflow procedures, any assignments to hidden variables that have been made by calls on descriptive procedures will cause adjustment to the corresponding variables H1, H2, H3, H4, H5, H6.

APPENDIX D.

PROCEDURES FOR PREPARING ALGOL PROGRAMS FOR COMPILATION AND EXECUTION

An ALGOL compilation will consist of either a block or a procedure declaration. In the first case, what is produced is a free-standing program which may call external procedures but which is assumed to operate otherwise as a free-standing program. In the second case the result is a separately compiled procedure which may be called by another program.

Users create programs by entering ALGOL statements into remote and local peripheral or terminal devices which are connected to a computer operating under the operating system.

Three modes of operation are available to the user: local batch, remote batch, and time-sharing. The only user differences among the three modes are the I/O device assignments for the system output and input files, the presence of necessary user-GCOS communication via control cards or command language, and the assumed compiler options for the compilation process.

BATCH MODE

In the local batch mode, the system I/O devices are the card reader card punch and line printer. The user communicates directly with the operating system for system services via the control cards and the usable slave mode instructions. The execution of user programs submitted via the local batch mode is carried out directly under the operating system. The user's program exists as a separate batch job. Input processing is performed by System Input.

The remote batch mode is equivalent to the local batch mode in capability. The only difference is the assignment of the system I/O device to the remote terminal as remote files (not direct access) rather than to the local card reader and local printer/punch.

BATCH CALL CARD

The system call card for ALGOL in batch mode is:

1	8	16 (operand field)
$	ALGOL	Options

Operand Field:

The operand field specifies the system options. The following options are available with ALGOL (standard batch options are underlined):

LSTIN - A listing of source input will be prepared by the ALGOL compiler.

NLSTIN - No listing of the source input will be prepared.

LSTOU - A listing of the compiled object program output will be prepared.

NLSTOU - No listing of the compiled object program output will be prepared.

DECK - A binary object program deck will be prepared as output.

NDECK - No binary object program deck will be prepared.

COMDK - A compressed source deck will be prepared as output.

NCOMDK - No compressed source deck will be prepared as output.

SYMTAB - A symbol cross reference report will be prepared as output.

DEBUG - Source line numbers will be printed with error messages at execution time.

NDEBUG - Source line numbers will not be printed with error messages at execution time.

DUMP - Slave core dump will be given if the compilation activity terminates abnormally.

NDUMP - Program registers upper SSA, and slave program prefix will be dumped if the compilation activity terminates abnormally.

SAMPLE BATCH DECK SETUP

The following are the required control cards for the compilation and execution of a batch ALGOL activity. The $ control cards are fully described in the Control Cards Reference Manual.

```
 1       8         16
 $       SNUMB
 $       IDENT
 $       OPTION    ALGOL
 $       ALGOL     Options
           .     }
           .     }          ALGOL Source Deck(s)
           .     }
 $       EXECUTE   Options
{$       FFILE                                  }
{$       < physical device assignment >         }
 $       ENDJOB
 ***EOF
```

The $ OPTION card with option ALGOL is required for every execution activity containing at least one deck produced by the ALGOL compiler. It must be the first card of the execution activity. Other options, as desired, may be used on this card but ALGOL is required.

The $ FFILE and physical device assignment cards are enclosed in braces to indicate that they may or may not be required for a particular activity. Cards of the form $ TAPE or $ FFILE option define physical devices which are to be associated with files referenced in the ALGOL program through calls on the input/output data transmission procedures. A card of this type is required for every referenced file other than 05 and 06 and input files produced with the $ DATA card.

The $ FFILE card provides fine control over the characteristics of each logical field. The $ FFILE card option DSTCOD (used only with programs generated by the ALGOL compiler) is of the form:

DSTCOD/(XXX)

where XXX may be any of the following:

Mnemonic	Device
PRNTR	Line Printer
BCRDR	Card Reader (Binary)
DCRDR	Card Reader (Decimal)
BCPNCH	Card Punch (Binary)
DCPNCH	Card Punch (Decimal)
MTAPE	Magnetic Tape
DISC	Disk
DRUM	Drum
PTMODS	5/6 channel paper tape
PTMODD	7/8 channel paper tape

The destination code subfield (DSTCOD) defines the type of logical device which is to be associated with a file, independent of the physical device on which the file may reside. In this way the system limits, i.e., P and P', for a file are defined. For example, to produce a listing file which must be saved on tape for future reference, a $ TAPE DSTCOD/(PRNTR) would define the output as destined for a line printer. In the absence of the $ FFILE card or the DSTCOD subfield the logical device will be assumed to be the same as the physical device.

TIME-SHARING OPERATION

Command Language

The standard means of communication with the Time-Sharing System (TSS) is by way of a teletypewriter used as a remote terminal. Other compatible devices may also be used, but use of a teletypewriter is assumed in this manual. The user may choose either the keyboard/printer or paper-tape teletypewriter unit for input/output, or combine both. In either case, the information transmitted to and from the system is displayed on the terminal-printer. Keyboard input will be used for purposes of description; instructions for the use of paper tape are given under "Paper Tape Input" in this appendix.

The user "controls" the time-sharing system primarily by means of a command language, a language distinct from any of the specialized programming languages that are recognized by the individual time-sharing compilers/processors (e.g., the Time-Sharing FORTRAN language). The command language is, for the most part, the same for users of any component of the time-sharing system; i.e., FORTRAN, BASIC, Text Editor, etc. A few of the commands pertain to only one or another of the component time-sharing systems, but the majority of them are, in form and meaning, common to all component systems.

The commands relate to the generation, modification, and disposition of program and data files, and program compilation/execution requests. The complete time-sharing command language is described in Time-Sharing System General Information Manual.

Once communication with the system has been established, any question or request from the system must be answered within ten minutes, except for the initial requests for user identification (user-ID) and sign-on password, which must be given within one minute. If these time limits are exceeded, the user's terminal will be disconnected.

Time-Sharing Commands for ALGOL

Valid time-sharing system commands for ALGOL are listed in Table D-1. These commands are fully described in the Time-Sharing System General Information Manual.

Table D-1. ALGOL Time-Sharing System Commands

Command	Applicable At Build Mode
ABC	Yes
ACCESS	Yes
ASCASC	Yes
ASCBCD	Yes
[a] AUTOMATIC	Yes
BCDASC	Yes
BPRINT	Yes
BPUNCH	Yes
BYE	Yes
CATALOG	Yes
[a] DELETE	Yes
DONE	Yes
EDIT	Yes
[a] ERASE	Yes
FDUMP	Yes
GET	Yes
HELP	Yes
HOLD	Yes
JABT	Yes
JOUT	Yes
JSTS	Yes
LENGTH	Yes
[a] LIB	Yes
LIST	Yes
[a] NEW	Yes
NEWUSER	Yes
NO PARITY	Yes
[a] OLD	Yes
PARITY	Yes
[a] PERM	Yes
[a] PRINT	Yes
[a] PURGE	Yes
RECOVER	Yes
#RECOVER	No
[a] RELEASE	Yes
REMOVE	Yes
[a] RESAVE	Yes
[a] RESEQUENCE	Yes
ROLLBACK	Yes
#ROLLBACK	No
[a] RUN	Yes
[a] SAVE	Yes
SCAN	Yes
SEND	Yes
STATUS	Yes
[a] SYSTEM	Yes
[a] TAPE	Yes

[a] not applicable at subsystem-selection level

Log-on Procedure

The user, to initiate communication with the time-sharing system, performs the following steps:

- Turns on the terminal unit
- Obtains a dial-tone
- Dials one of the numbers of his time-sharing center

The user will then receive either a high-pitched tone indicating that his terminal has been connected to the computer or a busy signal. The busy signal indicates, of course, that no free line is presently available.

Once the user's terminal has been connected to the computer, the time-sharing system begins the log-on procedure by transmitting the following message:

HIS SERIES 6000, SERIES 600 ON(date)AT(time)CHANNEL(nnnn)

where time is given in hours and thousandths of hours (hh.hhh), and nnnn is the user's line number.

Following this message, the system asks for the user's identification:

USER ID --

The user responds, on the same line, with the user-ID that has been assigned to him by the time-sharing installation management. This user-ID uniquely identifies a particular user already known to the system, for the purposes of locating his programs and files and accounting for his usage of the time-sharing resources allocated to him. An example request and response might be:

USER ID -- J.P.JONES

Note: A carriage return must be given following any complete response, command, or line of information typed by the user.

(The user's response is underlined here for illustration). After the user responds with his user-ID, the system asks for the sign-on password that was assigned to him along with his user-ID, as follows:

PASSWORD
XXXXXXXXXXXX

The user types his password directly on the "strikeover" mask provided below the request PASSWORD. The password is used by the system as a check on the legitimacy of the named user. The "strikeover" mask insures that the password, when typed, cannot be read by another person. (In the event that either the user-ID or password is twice given incorrectly, the user's terminal is immediately disconnected from the system.) At this point, if the accumulated charges for the user's past time-sharing usage equals or slightly exceeds 100 per cent of his current resource allocation, he will receive a warning message. If his accumulated charges exceeds 110 per cent of his current resources, he receives the message:

RESOURCES EXHAUSTED - CANNOT ACCEPT YOU

and his terminal is immediately disconnected. (The user may also receive the following information message if his situation warrants it:

n BLOCKS FILE SPACE AVAILABLE

(This condition does not affect the log-on procedure.)

Assuming that the user has responded with a legitimate user-ID and password and has not over-extended his resources, the time-sharing system then asks the user to select the processing system that he wants to work with; this is called the system-selection request. In this case, the user would respond with ALGOL:

```
SYSTEM ? ALGOL
```

The user is then asked whether he now wants to enter a new program (NEW) or if he wants to retrieve and work with a previously entered and saved program (OLD); the request message is:

```
OLD OR NEW -
```

If the user wishes to start a new program (i.e., build a new source file), he responds simply with:

```
NEW
```

If, on the other hand, he wants to recall an old source-program file, he responds with:

```
OLD filename
```

where filename is the name of the file on which the old program was saved during a previous session at the terminal (see the SAVE command).

Following either response, the system types the message "READY", returns the carriage, and prints an asterisk in the first character position of the next line:

```
READY
*
```

An example of a complete log-on procedure, up to the point where the ALGOL system is ready to accept program input or control commands, might be as follows:

```
HIS SERIES 6000, SERIES 600 ON 10/14/71 AT 14.768 CHANNEL 0012

USER ID - J.P.JONES
PASSWORD
XXXXXXXXXXXX      - (user's password is typed over the mask)
SYSTEM - ALGOL
OLD OR NEW - NEW  - (NEW is shown arbitrarily for
                    illustration)
READY
*                 - (the user begins entering input on this
                    line)
```

Entering Program - Statement Input

After the message:

```
READY
*
```

the system is in build-mode (as indicated by the initial asterisk) and is ready to accept ALGOL program-statement input or control commands. All lines of input other than control commands are accumulated on the user's current file. Normally the current file will be the file that contains the program he wants to compile and run at this session. If he is building a new file (NEW response to OLD OR NEW--), his current file will initially be empty. If he has recalled an old file (OLD filename) the content of the named old file will initially be on his current file, and any input typed by the user -- excepting control commands -- will be either added to, merged into, or will replace lines in the current file, depending upon the relative line numbering of the lines in the file and the new input. (This process is explained under the heading "Correcting or Modifying a Program," below.)

Following each line of noncommand-language input and the terminating carriage response, the system will supply another initial asterisk, indicating that it is ready to accept more input.

Format of Program - Statement Input

A line of ALGOL input -- as distinct from a control command -- can contain one of the following:

1. One or more ALGOL statements.
2. A partial statement.
3. A continuation of a statement left incomplete in the preceding line of input.
4. A comment.
5. A combination of (3) and (1) or (2), in that order.
6. A combination of (1) and (2).

A line of input must begin with a line-sequence number of from one to six numeric characters. The line-sequence number facilitates correction and modification of the source program (described below); hereinafter, the line-sequence number will be referred to simply as the "line number".

The line number is always terminated (immediately followed by) a non-numeric character. A line number must have a numeric value less than 2 (262144). If the user wants a numeric in column 1 of an ALGOL statement, a pound sign (#) must immediately follow the line number. An input line consisting only of numbers or of numbers followed by a pound sign will be ignored by the compiler. In order to insert blank lines into a program, the line number must be followed by at least one blank.

Correcting or Modifying a Program

Keyboard input is sent to the computer and written onto the user's current file in units of complete lines. A line of terminal input is terminated by a carriage return and no part of the line is transmitted to the system until that carriage return is given. Therefore, corrections or modifications can be done at the terminal at two distinct levels:

1. Correction of a line-in-progress (i.e., a partial line not yet terminated).

2. Correction or modification of the program (i.e., the contents of the user's current source file) by the replacement or deletion of lines contained therein, or the insertion of new lines.

The correction of a typing error that is detected by the user before the line is terminated can be done in one of two ways. He may delete one or more characters from the end of the partial line or he may cancel the incomplete line and start over. The rules are as follows:

a. Use of the commercial "at" character (@) deletes from the line the character preceding the @ character; use of n consecutive @ characters deletes the n preceding characters (including blanks.)

   Examples:

   *ABCDF@E would result in ABCDE being transmitted to the program file.

   *ABC⌀DEF@@@@GHJ would result in ABCGHJ being transmitted. (The characters to be deleted are underlined for illustration.)

b. Use of the CTRL (control) and X keys, depressed simultaneously, causes all of the line to be deleted. The characters DEL are printed to indicate deletion and the carriage is automatically returned. For example:

```
*ACDEFG   CTRL/X   DEL   (all characters deleted)
*                        (ready for new input)
```

Correction or modification of the current source file is done on the basis of line numbers and proceeds according to the following rules:

1. Replacement.A numbered line will replace any identically numbered line that was previously typed or contained on the current file (i.e., the last entered line numbered nnn will be the only line numbered nnn in the file).

2. Deletion. A "line" consisting of only a line number (i.e., nnn) will cause the deletion of any identically numbered line that was previously typed or contained on the current file.

3. Insertion. A line with a line-number value that falls between the line-number values of two pre-existing lines will be inserted in the file between those two lines. If the line number is less than the first line number it is inserted at the beginning of the file; if greater than the largest line number, it is inserted at the end of the file.

At any point in the process of entering program-statement input, the LIST command may be given, which results in a "clean", up-do-date copy of the current file being printed. In this way, the results of any previous corrections or modifications can be verified visually. Following the response (or command) OLD filename, the LIST command can be used initially to inspect the contents of the current source file (i.e., the "old" program).

The ALGOL Run Command

The RUN command has the form:

RUN H -nnn fs = fh; fc(options) flib # fe

where: RUN H — is the command RUN or RUN H. The latter form is used to display a heading line on the terminal giving date, time, and SNUMB.

-nnn — nnn is the maximum time in seconds of processor time, that the program is to be allowed for execution.

fs — is the set of file descriptors for source files in the standard BCD card image format, in compressed card image format (COMDK), or in time-sharing ASCII format and/or descriptors for binary card image object files. These files serve as inputs to the compiler and/or loader. Where a BCD or COMDK source file is supplied, fs may also include a descriptor for an alter file in BCD format. The alter file must begin with a $ UPDATE card and must be in alter number sequence. If there are many BCD or COMDK source files in the list, the alter file will update the first. Alternatively, the list fs may consist of a single file descriptor that points to a previously generated system loadable (H*) file.

A file descriptor consisting of the single character * indicates the current file (*SRC). The list is optional and, when missing, indicates that only the current file (*SRC) is to be compiled.

fh — is a single file descriptor of a random file into which the system loadable file produced by the General Loader will be saved if the compilation is successful. This file will be written if no fatal errors occur during compilation. If the named file does not exist, a permanent random file of 36 blocks will be created and added to the user's catalog. If the field is missing, the H* file is generated into a temporary file. The presence of this option is valid only when the program indicated by the list fs, the ALGOL library, and the user library (if any) is bindable (no outstanding SYMREFs). If the General Loader indicates that outstanding SYMREFs exist, an executable H* file will be created, but any reference to an unsatisfied SYMREF will cause the program execution to be abnormally terminated (the General Loader inserts a MME GEBORT at references to unsatisfied SYMREFs. When a MME is encountered during the execution of a time-sharing subsystem, GCOS and the Time-Sharing Executive simulate an illegal operation fault.)

= — must be included whenever fh, fc, (options), or flib options are desired.

fc — is a single file descriptor of a sequential file into which the compiler is to place the binary (C*) result of any indicated compilation(s). One object module is written to this file for each source program in the file(s) given by fs. If the named file does not exist, a permanent linked file of 3 blocks will be created and added to the user's catalog. This file will expand as necessary to hold the object decks. In this case the field fs plus the libraries need not indicate a complete program (individual or collections of routines may be compiled and saved). When this optional field is missing, a C* file will not be generated. When present, the DECK option is turned on for the compilation process.

(options) — is a list of options contained within parentheses and comma-separated. Some of these options affect the compilation process, and some the loading. The following compiler options are available for time-sharing; they are described under the $ ALGOL card; underlined is default.

DEBUG - Source line numbers will be printed with error messages at execution time.

NDEBUG - Source line numbers will not be printed.

The remaining options have to do with the loading process. The underlined option is the default case.

GO - The program will be executed at the completion of compilation.

NOGO - The program will not be executed at the completion of the compilation. If specified, the object program will be saved. If no object (H*) save file is specified, only the compilation will be performed (the General Loader will not be called).

ULIB - File descriptors exist following the end of the options field which locate user libraries which are to be searched for missing routines prior to searching for them in the system library.

NOLIB - No user libraries are to be used.

TIME=nnn - The batch compilation and/or General Loader activity time limits will be set to nnn seconds; where nnn $\leq$ 180. If not specified, nnn is set to 60.

CORE=nn - The batch compilation an/or General Loader activity core requirment will be set to nnK+9K or 23K, whichever is larger. If not specified, nn is set to 16.

URGC=nn - The urgency for the batch compilation and/or General Loader activity will be set to nn, where nn $\leq$ 40. If not specified, nn is set to 40.

TEST - A test version of the compiler and/or General Loader is to be used for the batch activity. There must be an accessed file (in the AFT) of the name ALGOL. If these two conditions are met, then file ALGOL will be allocated as file code ** in the batch activity.

REMO - Temporary files created for the batch process will be removed from the AFT as they are no longer needed. This option keeps the number of files in the AFT down to a minimum but causes more time to be spent processing each RUN command.

NAME=name - Provides a name for the main link of the saved H* file. May be used both at time of creation of this file and subsequently as it is used. This name is placed in the SAVE/field of the $ OPTION card.

flib is a sequence of file descriptors pointing to random files containing user libraries to be searched before the system library.

must be included whenever the fe option is desired.

fe is a set of file descriptors for files which will be required during execution. Each descriptor must specify a file name (or an alternate name, if necessary) of the form nn, where $01 \leq nn \leq 40$. This name represents a logical file code referenced by I/O statements in the program. I/O may be teletypewriter-directed by specifying a descriptor of the form "nn". File codes 05 and 06 are implicitly defined for teletypewriter I/O and need not be mentioned in the run command.

Log-Off Procedure

To terminate one's current session with the time-sharing system and disconnect the terminal, the BYE command may be given either in build-mode or at the subsystem-selection level:

```
*BYE
```

or

```
SYSTEM ? BYE
```

In either case, a report of the user's time-sharing usage charges is given, as illustrated by the following example, and the terminal is disconnected:

```
**RESOURCES USED $ 4.47, USED TO DATE $ 919.02= 92%

**TIME SHARING OFF AT 12.655 ON 10/14/71
```

Batch Activity Spawned by RUN

As an example of the simplest case, consider that some source file is current in *SRC, and a RUN command is typed with none of the optional fields. A job setup comparable to the following will be dispatched to the batch system.

```
$       SNUMB     nnnnT,40
$       USERID
$       IDENT
$       LOWLOAD   36
$       OPTION    ALGOL,NOMAP,NOGO,SAVE/OBJECT
$       USE       .TSGF
$       ALGOL     NLSTIN,NDECK
$       LIMITS    2,25K
$       FILE      S*,X1R          source file *SRC
$       FILE      P*,X2S          diagnostic report only
$       EXECUTE
$       FILE      P*,X2R
$       FILE      H*,X3R,3R       bound program
$       ENDJOB
```

The results of compilation and loading are returned on files P* and H*. P* is read and scanned for compiler and/or loader diagnostics. These are displayed on the teletypewriter and if there have been no fatal errors, the fully bound program will be loaded from H* and execution will proceed.

Supplying Direct-Mode Program Input

During program execution, keyboard input may need to be supplied to satisfy one or more input statements in the program. Each time input is required, the equal-sign character, "=", will be printed at the terminal. The user begins typing the input immediately following the equal sign. A carriage return signals the end of the input line.

An end-of-file condition will be generated when the equal sign is immediately followed by a file separator character (control, shift, and L keys depressed simultaneously), which must be immediately followed by a carriage return.

The ALGOL subroutine library allows multiple output statements to refer to the same output line; i.e., line spacing occurs only when the library encounters an alignment mark or when the line overflows. When both input from and output to the teletype are required, the user must insure that any complete lines which are to appear before an input line must be terminated with the proper alignment marks.

It is also possible to input data from a paper tape. The actual characters transmitted to the terminal from an input statement are: carriage return (CR), rubout (RO), equal sign (=), and sign-on (X-ON). The sign-on character activates the paper tape reader if the reader is in the ready state. A ready state is achieved by having the paper tape "loaded" and the reader switches set to "AUTO" or "TD-ON". Paper tapes which are to be used in this way should end each line with the characters: carriage return (CR), sign off (X-OFF), line feed (LF), rubout (RO). The sign-off character turns off the reader but leaves it in a ready state for any subsequent READ's. Paper tapes must begin with an appropriate leader of RO characters.

Emergency Termination of Execution

The use of the BREAK key will terminate program execution. All file output buffers, including the teletypewriter buffer, will be flushed, the files will be closed, and a NORMAL TERMINATION message will be generated. Control will return to the build-mode after the use of the BREAK key.

Paper Tape Input

In order to supply build-mode input from paper tape, the user gives the command TAPE. The system responds with READY. At this point, the user should position his tape in the reader and start the device. Input is terminated when either the end-of-tape occurs, the user turns off the reader, and X-OFF character is read by the paper tape reader, or a jammed tape causes a delay of over one second between the transmission of characters.

At present a maximum of 80 characters are permitted per line of paper tape input. Excessive lines will be truncated at 80 characters with the remaining data placed in the next line. A maximum of two disk links (7680 words) of paper tape input will be collected during a single input procedure. All data in excess of two disk links will be lost.

Example of a Time-Sharing Session

A comprehensive example of program creation, testing, correction and modification follows. Explanations are enclosed in parentheses; they are not part of the printout.

```
USER  ID -BALLARD
PASSWORD--
XXXXXXXXXXXX
SYSTEM ?ALGOL
OLD OR NEW - NEW
READY
*0010'BEGIN'
*0020'COMMENT'
*0030       THIS IS THE OLD GAME OF THE TOWERS OF HANOI.
*0040       A NUMBER OF DIFFERENT RINGS @@@@@@ SIZED RINGS ARE
                                          (typing correction)
*0050       TO BE MOVED FROM POLE 2@1 TO POLE 3 WITH
*0060       ASSISTANCE FROM POLE 2. ONE RING AT A TIME
*0070       IS MOVED AND A LARGER RING CAN NOT REST ON
*0070       IS MOVED AND A LARGER RING MAY NOT REST ON
                                            (replace line 70)
*0080       A SMALLER;
*0090#
*0100#
*0110    'PROCEDURE' HANOI(N,P1,P2,P3);
*0120    'INTEGER' N,P1,P2,P3;
*0130       'IF' N=0 'THEN' 'GOTO' EXIT;
*0140       HANOI(N-1,P1,P3,P2);
*0150       OUTPUT3(6,"/,"MOVE RING\,ZZD," FROM POLE\,ZD," TO POLE\,
*0160       ZD\,N,P1,P2);
*0170       HANOI(N-1,P3,P2,P1);
*0180 EXIT:
*0190    'END';
*0125    'BEGIN'
*0200#
*0210#
*0220    HANOI
*0220    INPUT1(5,"L@ ,N);
*0015'INTEGER' N; (insert line between lines 10 and 20)
*0230#
*0240    HANOI(N,1,3,2);
*0250#
*0260'END' OF ENTIRE PROGRAM
*0190    'END' OF PROCEDURE HANOI;
*LIST    (list corrected program)
```

```
0010'BEGIN'
0015'INTEGER' N;
0020'COMMENT'
0030       THIS IS THE OLD GAME OF THE TOWERS OF HANOI.
0040       A NUMBER OF DIFFERENT SIZED RINGS ARE
0050       TO BE MOVED FROM POLE 1 TO POLE 3 WITH
0060       ASSISTANCE FROM POLE 2. ONE RING AT A TIME
0070       IS MOVED AND A LARGER RING MAY NOT REST ON
0080       A SMALLER;
0090#
0100#
0110    'PROCEDURE' HANOI(N,P1,P2,P3);
0120    'INTEGER' N,P1,P2,P3;
0125    'BEGIN'
0130       'IF' N=0 'THEN' 'GOTO' EXIT;
0140       HANOI(N-1,P1,P3,P2);
0150       OUTPUT3(6,"/,"MOVE RING\,ZZD," FROM POLE\,ZD," TO POLE\,
0160       ZD\,N,P2,P2);
0170       HANOI(N-1,P3,P2,P1);
0180 EXIT:
0190    'END' OF PROCEDURE HANOI;
0200#
0210#
0220    INPUT1(5,"  ,N);
0230#
0240    HANOI(N,1,3,2);
0250#
0260'END' OF ENTIRE PROGRAM

READY

*SAVE ALGTEST (save program)
DATA SAVED--ALGTEST
*RUN (run program)
NO ERROR IN ABOVE COMPILATION
15098 WORDS WERE USED FOR THIS COMPILATION
=3   (type input data)
MOVE RING 1 FROM POLE 1 TO POLE 3
MOVE RING 2 FROM POLE 1 TO POLE 2
MOVE RING 1 FROM POLE 3 TO POLE 2
MOVE RING 3 FROM POLE 1 TO POLE 3
MOVE RING 1 FROM POLE 2 TO POLE 1
MOVE RING 2 FROM POLE 2 TO POLE 3
MOVE RING 1 FROM POLE 1 TO POLE 3
  7275(10) WAS THE HIGHEST USED MEMORY LOCATION.

NORMAL TERMINATION
```

REMOTE BATCH INTERFACE

Refer to the GRTS Programming Reference Manual, for a description of the deck setup required for submitting a batch job from a remote terminal.

FILE SYSTEM INTERFACE

The File System provides multiprocessor access to a common data base. The file system allocates permanent file space and controls file access for users in local and remote batch and time-sharing. The file system is fully described in the manual File System.

TERMINAL/BATCH INTERFACE

The CARDIN time-sharing subsystem allows the user to submit a batch job from a time-sharing terminal. This capability is fully described in the manual Time-Sharing System Terminal/Batch Interface Facility.

FILE FORMATS

The compiler will accept either ASCII or BCD source input. ASCII source input must be in the TSS format (media code 5) and is assumed to have line numbers. An ASCII input line of only numbers (or only numbers followed by a pound sign) will be ignored. In order to insert blank lines into the program, the line number must be followed by at least one blank.

The subroutine library input routines will accept either ASCII or BCD input. ASCII input may be either in the TSS format (media code 5) or in "standard ASCII" format (media code 6). If an input file is in TSS format, it is assumed to have line numbers. The line numbers will be removed by the input routines.

The subroutine library will write either ASCII or BCD output files. ASCII output files will be in the "standard ASCII" format (media code 6). Each output line will have one line feed appended to the end of the record. The last word of a line will be filled with "rub-out" characters. The alignment mark for restoring the page (↑) will generate a record of 16 consecutive line feeds.

In the batch mode of operation, file codes 5 and 6 are assumed to be in BCD format. File code 5 is for the card reader (I*) and file code 6 is for the printer (P*). If I/O is to be performed on a file which is (or will be) in ASCII format, a statement SYSPARAM (FC, 24, 4) must precede the first reference to that file.

In the time-sharing mode of operation, file codes 5 and 6 are assumed to reference the teletype (i.e. in ASCII format). Since the destination code (DSTCOD) capability is not available in time-sharing, if I/O is to be performed on a file other than 5 or 6, a call to SYSPARAM with the appropriate parameters must precede the first reference to that file.

APPENDIX E

BASIC SYMBOLS WITH EQUIVALENT INTERNAL INTEGER VALUES

SYMBOL	VALUE	SYMBOL	VALUE
A	32	'GQ'	163
B	33	'GR'	164
C	34	'NQ'	159
D	35	'EQV'	154
E	36	'IMP'	155
F	37	'OR'	156
G	38	'AND'	157
H	39	'NOT'	158
I	40	.	511
J	41	,	172
K	42	:	178
L	43	;	176
M	44	(	152
N	45	)	174
O	46	[	153
P	47	]	173
Q	48	"	182
R	49	\	183
S	50	'	510
T	51	←	144
U	52	∅	181
V	53	'ARRAY'	134
W	54	'BEGIN'	128
X	55	'BOOLEAN'	133
Y	56	'CODE'	140
Z	57	'COMMENT'	180
0	0	'DO'	148
1	1	'ELSE'	151
2	2	'END'	175
3	3	'EXTENDED REAL'	131
4	4	'FOR'	143
5	5	'GO TO'	142
6	6	'IF'	149
7	7	'INTEGER'	132
8	8	'LABEL'	138
9	9	'NONLOCAL'	141
'TRUE	509	'OWN'	129
'FALSE'	508	'PROCEDURE'	135
+	165	'REAL'	130
-	167	'RENAME'	184
*	168	'STEP'	145
/	169	'STRING'	139
%	170	'SWITCH'	136
↑	171	'THEN'	150
'LS'	160	'UNTIL'	146
'LQ'	161	'VALUE'	137
'EQ'	162	'WHILE'	147

APPENDIX F

STACK TRACING ROUTINE

PURPOSE

To print the stack and non-procedural variables and/or to give an octal dump when an error is detected by the ALGOL run-time routines.

USAGE

The action to be taken is governed by the options on a $ USE control card. The two options are .ADUMP and .ADUMR.

The option .ADUMP gives a decimal dump when the first error is detected. Also it gives a dump when the number of errors detected exceeds the maximum number allowed (at which time the execution is terminated). It gives a dump when the program is aborted by GCOS.

The option .ADUMR gives an octal dump of all of memory when the first error is detected.

The $ USE card is placed after the $ OPTION card and preceding the program. Both .ADUMP and .ADUMR may be used in the same execution activity.

APPENDIX G

ALTERNATE SYMBOL REPRESENTATIONS

SYMBOL	ALTERNATE(S)
'LS'	<
'EQ'	=
'GR'	>
'GQ'	>=
'LQ'	<=
'NQ'	/=
←	:= or .=
:	..
;	.'
[	(/
]	/)
\	'('
%	')'
↑	//
	'POWER' or **

APPENDIX H

THE % OPTION CARD

A special internal directive card is recognized by the ALGOL compiler. It is indicated by the percent sign (%) in column 1 and blanks in columns 2-6.

Options begin in column 7 (or later) and are separated by commas. Currently the following options are available:

NLSTIN	Suppress listing of the source program beginning with the next source card.
LSTIN	Resume listing of the source program.
NXREF	Suppress symbol cross reference information beginning with the next source card.
XREF	Resume symbol cross reference information.
LKDATA	A binary deck of the link preset data will be punched.
BOUND	Perform array subscript checking.
NBOUND	Discontinue array subscript checking.
LSTOU	Start producing an output listing.
NLSTOU	Discontinue producing an output listing.
NAPOST	Allows ALGOL basic symbol words to be blank delineated instead of apostrophe delineated.
APOST	Resume accepting only apostrophe delineated basic symbol words.

The LSTIN directive is inoperable unless the LSTIN option is in effect from the $ ALGOL control card. The NLSTIN/LSTIN options allow listing only selected portions of the source program.

The use of the XREF/NXREF options allow obtaining only selected portions of the symbol cross reference table. They are not dependent on the SYMTAB option on the $ ALGOL control card.

The use of the LKDATA option causes the compiler to generate a binary deck containing the link preset data which would normally be written on the LF file.

The BOUND option causes the compiler to generate coding which will test the value of each subscript in an array reference. If the value is within the range, defined by the lower and upper bounds in the array declaration, no action is taken. If the value is too small, the error message

<<< SUBSCRIPT #XX IS TOO SMALL. LOWER BOUND USED. ALTER YYYYY >>>

is printed and the value of the subscript will be replaced by the value of the lower bound. If the value of the subscript is too large, the error message

<<< SUBSCRIPT #XX IS TOO LARGE. UPPER BOUND USED. ALTER YYYYY >>>

is printed and the value of the upper bound is used as the subscript. The option NBOUND causes the compiler to stop generating coding for checking subscripts. The BOUND/NBOUND options allow checking the value of subscripts in only selected portions of a program.

The LSTOU/NLSTOU options are the same as used for the $ ALGOL control card. However, when used on the % OPTION card, an output listing of selected portions of the program is permitted.

The NAPOST option allows the ALGOL basic symbol words such as BEGIN and END to be blank-separated instead of apostrophe-separated. The following rules govern the recognition of these words by the compiler.

Blank Delineated	Apostrophe Delineated
Preceded by at least one character.	Preceded by one and only one apostrophe.
Followed by at least one character.	Followed by one and only one apostrophe.
Two basic symbol words need only one blank character between them (ØBEGINØINTEGERØ).	Two basic symbol words need two apostrophes between them - one trailing and one preceding ('BEGIN' 'INTEGER').
Neither the seimicolon, the end of card, nor any special character will be preceding or trailing blank charter.	Neither the semicolon, the end of card, nor any special character will be accepted in lieu of the preceding or trailing apostrophe.
The basic symbol must be completely spelled out.	Only the first three characters are used to compare (e.g., 'BEG' will be accepted as 'BEGIN').
No blanks are permitted between characters of a word (ØBOTOØ may not be written as ØGOØTOØ).	Blanks are not significant.

INDEX

Honeywell Bull

HONEYWELL INFORMATION SYSTEMS

BS 11, Rev. 0

Ref.: 19.30.220 A