3100
3200
3300
3500

COMPUTER SYSTEMS
ALGOL
REFERENCE MANUAL

CONTROL DATA
CORPORATION

Additional copies of this manual may be obtained from the nearest Control Data Corporation Sales office listed on the back cover.

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

February, 1966
Pub. No. 60134800

CONTENTS

INTRODUCTION

The ALGOL system described in this manual consists of the ALGOL programming language and a compiler for translating ALGOL programs into machine language for execution on the Control Data® 3100, 3200, 3300 and 3500 computers.

This reference manual presents the details and rules involved in writing a program; it also includes sufficient information to prepare, compile and execute such a program. Throughout this manual the name ALGOL means 3100/3200/3300/3500 ALGOL unless otherwise specified.

The 3100/3200/3300/3500 ALGOL language closely conforms to the language defined in the ALGOL-60 Revised Report[1]; the input-output procedures provided as part of the language conform closely to the set recommended by the ACM[2]. This manual therefore uses both the report and the ACM proposal as its basic reference material.

The terms used to describe 3100/3200/3300/3500 ALGOL are, wherever possible, the ones which have explicit meanings within the context of the ALGOL-60 Revised Report, the ACM input-output proposal, or general ALGOL literature. No attempt is made to explain or define ALGOL concepts except where it is necessary for an understanding of 3100/3200/3300/3500 ALGOL.

The reader is assumed to be familiar with the referenced documents and the Control Data publications: 3100/3200/3300 ALGOL General Information Manual, 3100/3200/3300 ALGOL Instant Manual, and the SCOPE/COMPASS Reference Manual.

In addition, the reader is referred to the following representative bibliography which, it should be noted, is not meant to be exhaustive:

Baumann, R., Feliciano, M., Bauer, F. L., Samelson, K.: Introduction to ALGOL, Prentice-Hall, Inc., 1964.

Dijkstra, E.W.: A Primer of ALGOL-60 Programming, Academic Press, 1962.

McCracken, Daniel D.: A Guide to ALGOL Programming, John Wiley & Sons, Inc., 1962.

1 The Communications of the ACM, 1963, vol. 6, No. 1, pp 1-17
Numerische Mathematik, Vol. 4, pp 420-453 (1963)
The Journal of the British Computer Soc.

2 International Organization for Standardization, Draft Proposal on the Algorithmic Language ALGOL, Appendix E:
"Proposal for Input-Output Procedures for ALGOL 60 (ACM)".
This proposal is a revision of:
"A Proposal for Input-Output Procedures for ALGOL 60" by Knuth et al.;
Communications of the ACM, vol. 7, No. 5, May 1964.
The revised proposal includes as a subset the report issued by the International Federation for Information Processing (IFIP):
Communications of the ACM, vol. 7, No. 10, Oct. 1964, pp 628-630.

The revised proposal, called the ACM proposal throughout this manual, is obtainable from:
American Standards Association Incorporated, 10 East 40th St.,
New York 16, N.Y.

GENERAL DESCRIPTION 1

COMPILER FEATURES

The ALGOL compiler for the 3100/3200/3300/3500 computers is based in design on the ALGOL compiler developed by Regnecentralen, Copenhagen, Denmark, for the GIER computer. This design was adopted and, to some degree, extended by Control Data to provide the most generally advantageous features for an ALGOL compiler.

These include the implementation of the complete ALGOL-60 language (wherever feasible and not in conflict with other advantages); comprehensive input-output procedures; extensive compile-time and object-time diagnostics; fast compilation; and a wide variety of compilation options, such as the ability to compile both ALGOL programs and ALGOL procedures; and the ability to generate and execute the object program in either normal or segmented form.

The compiler is designed to run under control of the SCOPE monitor on a 3100, 3200, 3300 or 3500 computer with a minimum of 8K words of memory storage. The compiler takes advantage of extra computer memory to compile larger programs and uses extra storage for intermediate information, thus reducing or eliminating references to scratch units.

COMPILER STRUCTURE

The ALGOL compiler is a four-pass system in which each pass performs a separate function in the total translation process; the output from one pass serves as input to the next.

The first three passes perform all of the syntactic and semantic analysis of the source text. Their output is either a list of encoded error messages or the object code in a special macro format.

The fourth pass either decodes the error messages or produces the various outputs from the compiler, including the object code in binary form. The object code may be requested in the normal form for loading and executing under control of the SCOPE monitor, or it may be requested in the segmented form.

A special routine, nominally pass 5 (although it takes no part in the actual compilation process), controls the loading and executing of an object program in the segmented form (Chapter 6). Execution in segmented mode can be performed as part of the same compilation or as a completely separate process.

LANGUAGE CONVENTIONS

ALGOL is described in terms of three languages in this manual: reference, hardware, and publication language.

The reference language is that in which ALGOL is defined in the ALGOL-60 Revised Report; it is computer independent and utilizes the basic ALGOL symbols, such as begin and end, to define the language syntax and semantics.

The hardware language is the representation of ALGOL symbols in characters that are acceptable to the computer; this is the language used by the programmer. For example, where the reference language calls for the use of the basic ALGOL symbol begin, the ALGOL programmer includes the seven hardware characters 'BEGIN' in his program. The hardware representations of ALGOL symbols are shown in Appendix A.

Unless otherwise stated or implied, the basic ALGOL symbols (reference language) rather than their character equivalents (hardware language) are used consistently throughout this manual. This convention simplifies the explicit and implicit references to the ALGOL language as defined in the ALGOL-60 Revised Report.

For publication purposes only, the underlining convention delineates the basic ALGOL symbols. These symbols are understood to have no relation to the individual letters of which they are composed. Other than this convention, the publication language is not considered in this manual.

SOURCE INPUT FORMAT 2

SOURCE DECK

The source deck must be in the form of cards or card images, described here only as cards. Each character of the source input is punched in one card column. A blank column has no effect on the source text, except in strings (Chapter 4). Blanks may be freely used, however, to facilitate reading.

Only columns 1-72 of each card are interpreted by the compiler; there is no syntactic meaning attached to these boundaries; any language structure may appear across the boundaries of two or more cards.

Each card is counted at compile time and assigned a line count (beginning at 0) for reference by error messages. This line count is included in any source listing requested, as are columns 73-80 of each card.

The compiler can compile either an ALGOL program or an ALGOL procedure; the user indicates on the control card which type of input is to be compiled. A separately compiled procedure is incorporated into a main program at the object program level (Chapter 7).

PROGRAM COMPILATION

Compilation (generation of object code) of an ALGOL program starts with the first ALGOL symbol begin ('BEGIN') in the source deck and terminates with the last ALGOL symbol end ('END'). Any information in the source deck prior to the first begin or following the final end is treated as a commentary. Comments are printed as part of the source listing and are included in the line count. Information in columns 1-8 of the first commentary card is treated as the program name and printed on the page headings of the source listing. If there is no initial commentary, the program name generated is XXXALGOL. The end of the source deck is indicated by the characters 'EOP' in card columns 10-14.

The program should be self-contained in that it makes no reference to variables not defined within it. If it contains a reference to a procedure which is to be compiled separately, it must include a procedure declaration and a code symbol and code number to replace the body (Section 5.4.6 Chapter 3).

PROCEDURE COMPILATION

The source deck rules for compilation of an ALGOL procedure are the same as described for an ALGOL program, except that compilation starts with the first ALGOL symbol procedure ('PROCEDURE') encountered, rather than begin. The procedure symbol may be preceded by one of the ALGOL symbols, real ('REAL'), integer ('INTEGER'), or Boolean ('BOOLEAN').

The procedure should conform exactly to the rules for an ALGOL procedure specified in a main ALGOL program. However, the procedure should be self-contained both in the sense described above for a program and in that it should not declare any own variables nor should it call itself. In all other aspects, any of the permitted facilities for describing an ALGOL procedure may be used.

The information in columns 1-8 of the first commentary card of a pre-compiled procedure is not used as the procedure name. Instead, the procedure name is CDPxxxxx where xxxxx is the number assigned to the procedure on the ALGOL control card and associated with the procedure by the code symbol. If all 5 digits of the number are not specified, it is zero-filled on the left. For example, the number 20 becomes 00020.

The following pages show the ALGOL coding form and the structure of the source decks for the two compilation modes.

ALGOL CODING FORM

CONTROL DATA CORPORATION

PROGRAM:

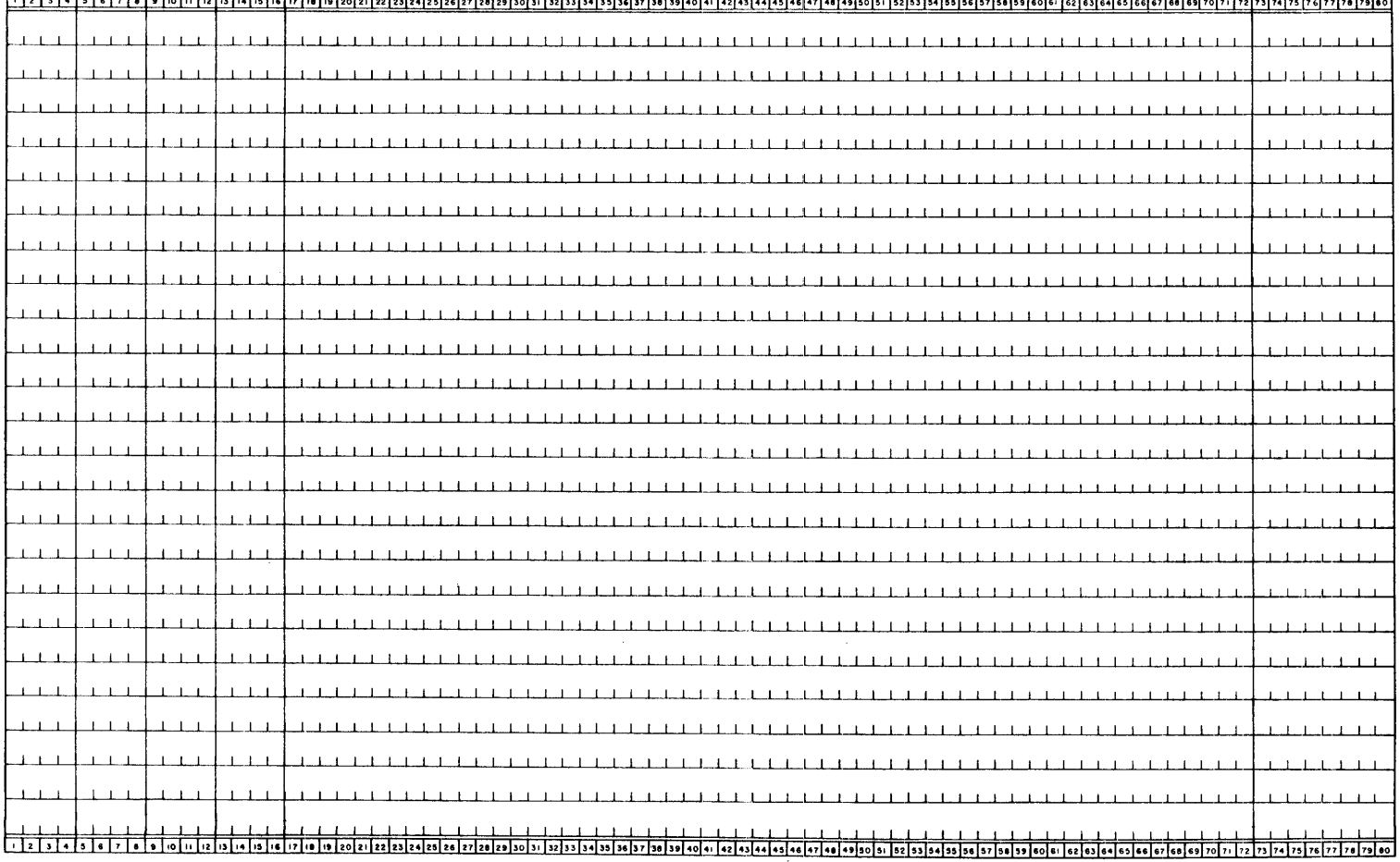
ROUTINE:

NAME

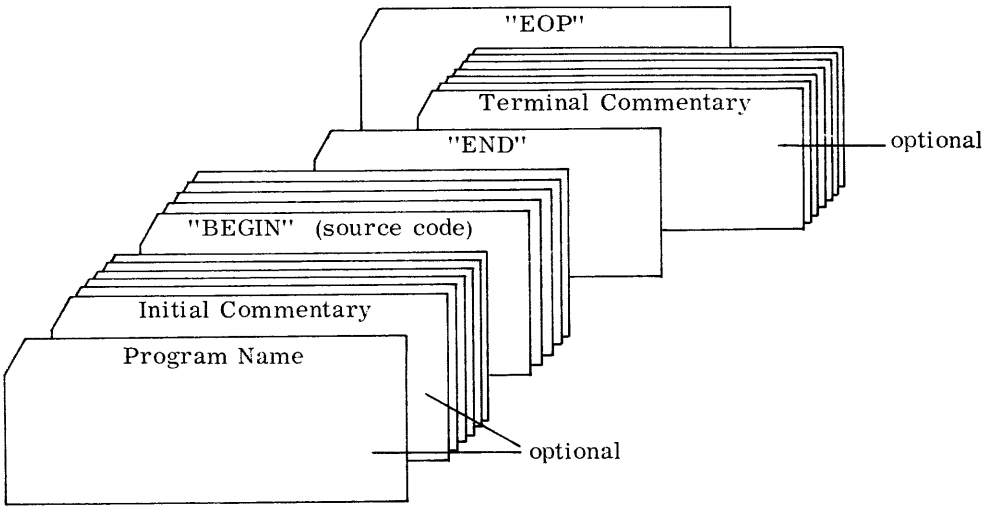
PAGE

DATE

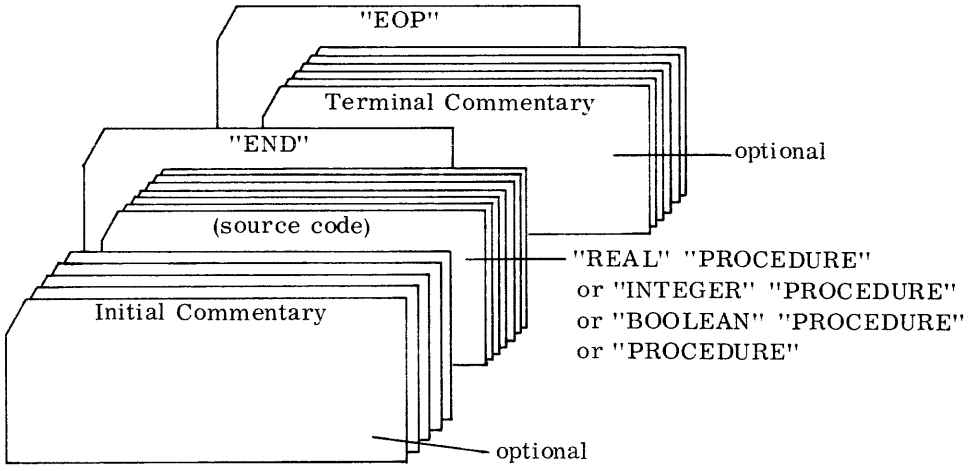
STATEMENTS | IDENT



Source Deck for Program



Source Deck for Procedure



To facilitate cross-referencing, the same numbering system is used throughout the present chapter as in the ALGOL-60 Revised Report. Since comments are made only on selected features, the section numbers do not run consecutively.

All descriptions of language modifications are made at the main reference in the report; wherever feasible, all other references are also noted. The reader should assume, however, that such modifications apply to all references to the features, noted or otherwise.

A section or feature not mentioned in this chapter is implemented fully in 3100/3200/3300/3500 ALGOL in exact accordance with the report. In addition to the language descriptions in this chapter, a set of reserved identifiers which reference input/output procedures are described in Chapter 4.

BASIC CONCEPTS

2. Basic Symbols, Identifiers, Numbers, and Strings, Basic Concepts

2.1 Letters
Since there is hardware representation for upper case letters only, lower case letters have no meaning in 3100/3200/3300/3500 ALGOL.

2.3 Delimiters
3100/3200/3300/3500 ALGOL contains two extra delimiters:

<code procedure body indicator> ::= code

<segment control indicator> ::= segment

The symbol code is included to permit reference to separately compiled procedures (section 5.4.6).

The symbol segment may appear only within a comment, so that it has no effect on compatibility with the ALGOL-60 Revised Report; nor does it prevent the program from being compiled on another ALGOL compiler. This symbol has no effect on the compilation process; it merely controls segmenting of the object program (Chapter 6).

At the point in the object program corresponding to the appearance of the segment symbol in the source program, the current segment is terminated and a new segment is begun.

Segmentation control increases efficiency of object program execution. For example, if the code generated for a for statement overlaps two segments; but can, in fact, fit into one segment, jumping between the segments is superfluous. The user can force the complete for statement code into a single segment by including the segment symbol just prior to the for statement in the source program; in this case, the for statement code begins the new segment.

The symbol comment and the whole of the comment feature is implemented exactly as in the report (Section 2.3) except that the symbol segment in the middle of a comment is recognized and treated as described above.

2.4 Identifiers

The maximum size of an identifier is 256 hardware characters. If a longer identifier is specified, only the first 256 characters are used.

The number of identifiers that can be handled by the compiler depends on their sizes and the memory capacity of the computer. With the largest size memory, the maximum is 4095 differently-spelled identifiers.

2.5 Numbers

A number can contain at most 14 decimal digits (leading zeros not counted), plus a decimal point, plus an exponent part as defined in the report. The maximum number of decimal digits in an exponent part is three.

2.5.4 Types

Variables of type integer are represented in 48-bit fixed-point form in the range:

$$-140,737,488,355,328 = -2^{47} < \text{integer} < +2^{47} = +140,737,488,355,328$$

Variables of type real are represented in normal floating-point form with a 36-bit mantissa, sign bit, and 11-bit exponent part in the range:

$$-10\uparrow 308 \doteq -2\uparrow 1023 < \text{real} < +2\uparrow 1023 \doteq +10\uparrow 308$$

Conversions between real and integer values are performed at both compile time and object time by closed subroutines.

All integer numbers with 14 or fewer digits are initially stored during compilation in fixed-point form. All integer numbers with more than 14 digits are converted to normal floating-point form. Numbers with a decimal point and/or an exponent part are converted to normal floating-point form. In all numbers, any digits after the fourteenth are treated as zeros (powers of ten).

If, in the source program, the number is involved directly with a variable, it is converted, if necessary, from its current form to normal floating-point if the variable is real or fixed-point form if it is integer.

If the number is not involved directly with a variable at compile time, it is left in its initial form; at object time, it is treated as a real variable if in normal floating-point form and as an integer variable if in fixed-point form.

2.6 Strings

2.6.1 Syntax
A proper string is defined in ALGOL as follows:

<proper string> ::= <any sequence of basic 6-bit BCD characters except 12_8>

2.6.3 Semantics
The string quotes 'and' are introduced to enable the ALGOL language to handle arbitrary sequences of basic characters.

EXPRESSIONS

3. Expressions

3.1.4 Subscripts

3.1.4.2 Subscript values are assumed to be in the range:

$-2\uparrow 23 <$ subscript value $< +2\uparrow 23$

No check is made to determine if a value is outside of this range. The upper 24 bits of the subscript integer variable are assumed to be zero and ignored.

The only subscript check is on the final address computed from the subscripts of the array. This check ensures that the address lies within the boundaries of the complete array; individual subscripts are not checked. The array bounds check may be suppressed throughout the object code with the control card option N (Chapter 7).

3.2.4 Standard Functions

When either of the standard functions ABS or SIGN is called directly, the object for it is generated in-line. A procedure call is generated only when ABS or SIGN is used as an argument to another procedure.

The algorithms for calculating the standard functions SIN, COS, ARCTAN, EXP, LN, and SQRT incorporated in the ALGOL system all yield results which have an accuracy of one part in $10\uparrow 10$.

All input-output functions are expressed as calls of standard procedures (described in Chapter 4). The list of reserved identifiers is expanded to include the names of these procedures, as follows:

IN LIST	HLIM	ARTHOFLW
OUT LIST	VLIM	PARITY
INPUT	HEND	EOF
OUTPUT	VEND	REWIND
IN REAL	NODATA	UNLOAD
OUT REAL	TABULATION	SKIPF
IN ARRAY	FORMAT	SKIPB
OUT ARRAY	SYSPARAM	ENDFILE
IN CHARACTER	EQUIV	BACKSPACE
OUT CHARACTER	STRING ELEMENT	IOLTH
GET ARRAY	CHLENGTH	MODE
PUT ARRAY	MANINT	

Calls to all of the standard procedures (both input-output and function) conform to the syntax of calls to declared procedures (Section 4.7.1) and in all other respects are equivalent to regular procedure calls. This specifically includes the activation of a standard procedure when its identifier appears as an actual parameter in a procedure call.

If a standard procedure is not needed throughout the entirety of a program, its identifier may be declared to have another meaning at any particular level; whenever it is used at that level, the identifier assumes the new meaning rather than that of the standard procedure.

3.3.4 Operations and Types
Arithmetic expressions whose type cannot be determined at compile time are considered real. For example, the parenthesized expression in the following statement is considered real if one or both of the arithmetic expressions R and S is real.

P (if Q then R else S)

3.3.4.3
The rule for evaluating an expression of the form A↑I is: The result is real - except where A is integer and I is a constant with positive integral value, in which case the result is integer. This differs from the report only when the base A is integer and the exponent I is an integer variable whose value is positive.

3.5.5 Unsigned integers as labels are not permitted.

STATEMENTS

4. Statements

4.1 Blocks
Blocks may be nested to a maximum of 32 levels.

4.3.5
If a goto statement is executed for a switch designator whose value is not in the range of the switch, the object program is terminated abnormally.

4.6.3
In a for statement, if the controlled variable is subscripted, the same array element is used as the control variable throughout the execution of the for statement, regardless of any changes that might occur to the value of the subscript expressions. The element used is the one referenced by the value of the subscript expressions on entry to the for statement.

4.7 Procedure Statements

4.7.5 Restrictions
The largest number of parameters which may be specified in a procedure call is 63; the largest number of constants is 62.

4.7.8
The symbol code is included to permit reference to separately compiled procedures (Section 5.4.6).

DECLARATIONS

5. Declarations
Because of their assigned positions in the object program stack (Appendix C), own variables are treated in definition as being global to the whole program. In the same way as other variables, however, they are treated as being local in scope.

5.2.2
own arrays with dynamic bounds (bounds which are not constants in the program) are not permitted. The following statement is illegal.

```
own integer array A [if C < 0 then 2 else 1:20]
```

5.4 Procedure Declarations

5.4.3 Semantics
The largest number of formal parameters that can be declared for a procedure is 63.

5.4.5 Specifications
The last sentence should be changed to read: "Specifications of all formal parameters, if any, must be supplied."

5.4.6 Code as Procedure Body
ALGOL source programs and procedures must be self-contained. A procedure declaration must be included for any procedure to be compiled separately from the program or procedure which references it. This declaration consists of a procedure heading plus a code procedure body.

The identifying name included in the heading need not be the same as the one specified when the procedure is compiled separately, nor need the names of the formals (though their number must be the same). The value part, if any, and the specification part may be omitted. All references to the replaced procedure must use the name declared in the procedure heading

included rather than the procedure heading when it is compiled separately.[1]

The code procedure body consists of the symbol code followed by a number xxxxx in the range 0-99999. This number must be assigned to the procedure when it is compiled separately[1] (Chapter 7).

In the following example, both AVERAGE and SQUAREAVERAGE can be replaced; the resulting program is shown below the original.

```
begin
      real procedure AVERAGE (LOWER, UPPER);
           value LOWER, UPPER;
           real LOWER, UPPER;
           begin AVERAGE:= (LOWER + UPPER)/2;
           end;
      real procedure SQUAREAVERAGE (LOW, HIGH);
           value LOW , HIGH;
           real LOW, HIGH;
           begin SQUAREAVERAGE:= SQRT (LOW↑2/4 + HIGH↑2/4);
           end;
      real X, Y, S, SQ;
      S:=0;
      SQ:=0;
      for X:=1 step 1 until 100 do
           begin Y:=X+1;
                 S:=S + AVERAGE (X, Y);
                 SQ:= SQ + SQUAREAVERAGE (X, Y);
           end;

end
```

1 Since the separately-compiled procedure is included in the main program at object program level, it does not have to be defined as an ALGOL procedure to produce the desired object code. The procedure can be generated in any way, provided the object code conforms to that produced by compilation of an ALGOL procedure.

```
begin
      real procedure MEAN (A, B);
           code 129;
      real procedure SQUAREAVERAGE (LOW, HIGH);
           value LOW, HIGH;
           real LOW, HIGH;
           code 527;
      real X, Y, S, SQ;
      S:=0;
      SQ:=0;
      for X:=1 step 1 until 100 do
           begin
                Y:=X+1;
                S:=S + MEAN (X, Y);
                SQ:=SQ + SQUAREAVERAGE (X, Y);
            end;

end
```

The first procedure body has been replaced by the symbol code with the identifying number 129. In the heading, the identifying name AVERAGE has been changed to MEAN, the formal parameter names to A and B, and the value and specification parts omitted. Reference to this procedure is to the name MEAN. The procedure called AVERAGE should be compiled separately with the code number 129 associated with it on the ALGOL control card.

The source deck for this compilation is the ALGOL control card followed by:

```
real procedure AVERAGE (LOWER, UPPER);
     value LOWER, UPPER;
     real LOWER, UPPER;
     begin
          AVERAGE:= (LOWER+UPPER)/2;
     end
```

and then the 'EOP' indication in columns 10 through 14 of the next card.

The second procedure body has been replaced by the symbol code and the identifying number 527. Since the procedure heading remains in identical form, the procedure is referenced exactly as before. The procedure called SQUAREAVERAGE should be compiled separately with the code number 527 associated with it on the ALGOL control card.

INPUT-OUTPUT 4

COMPARISON WITH ACM PROPOSAL

The following descriptions explain the differences between the input-output procedures included in this version of ALGOL and the procedures defined in the ACM proposal. To facilitate cross referencing, the same numbering system is used in this chapter as in the proposal. Since comments are made only on selected features, section numbers do not run consecutively. The ACM proposal is a continuation of the ALGOL-60 Revised Report, and begins with Section 6.

All descriptions of the modifications to the input-output procedures are made at the main reference in the proposal; wherever feasible, all other references are also noted. The reader should assume, however, that such modifications apply to all references to the features, noted or otherwise.

A section or feature not mentioned in this chapter is implemented fully in this version of ALGOL in exact accordance with the proposal.

This chapter also contains descriptions of additional input-output procedures which are not defined in the ACM proposal, and a description of the transmission error, end-of-file, and end-of-tape functions automatically supplied within the framework of the input-output procedures.

FORMATS

6. Formats

6.1 Number formats

6.1.1 Syntax
The structure <replicator> <string> is not permitted as an insertion component of a number format (Section 6.1.3.2).

6.1.3 Semantics

6.1.3.1 Replicators
A replicator of value 0 (either n or X) implies the absence of the quantity to which the replicator refers. The maximum size of a replicator is 32,766.

6.1.3.2 Insertions
String quotes 'and' are not allowed within a string inserted in a number format. The structure<replicator><string> is not permitted as an insertion component of a number format (Section 6.1.1).

6.1.3.3 Sign and Zero Suppression
On input, if no sign appears in the format and the number is negative, an error message is issued and the object program terminates abnormally. Output uses the standard format bounded on either side by an asterisk (Section 6.2.3.7).

6.1.3.5 Truncation
On output, the same number of significant digits will appear for a real number corresponding to the storage of the number in a 48-bit floating-point form (Section 2.5.4, Chapter 3). Either 10 or 11 significant digits are output, followed by trailing zeros, if necessary.

The letter T has no meaning when applied to an integer number and is ignored.

6.1.3.7 Two types of Numeric Format
The maximum number of D's and Z's appearing before the exponent part in a number format is 24; the maximum number of D's and Z's in the exponent part is 4. On output overflow the standard format bounded on either side by an asterisk is used (Section 6.2.3.7).

6.1.3.8 Input
If the input data does not conform to the format, an error message is issued, and the object program terminates abnormally.

6.2 Other Formats

6.2.1 Syntax
The character M has been added to the <nonformat> codes.

After each quantity of a format item is expanded by the corresponding replicator, the maximum length is 136 characters; the expanded format item corresponds to data on the external medium.

6.2.3.1 String Format

Because of the difference in the definition of a string (Section 2.6.1, Chapter 3), each of the S-positions in the format corresponds to a single basic character in the output string rather than a single basic symbol. If the string exceeds the number of S's, the leftmost basic characters are transferred; if the string is shorter, blank characters are filled to the right.

The string quotes ' and ' are represented internally by 1201_8 and 1202_8; string quotes contained in an output string will therefore print as :1 and :2; if written on magnetic tape in even parity, they can be subsequently recognized only as 0001_8 and 0002_8.

6.2.3.2 Alpha Format

Because of the difference in the definition of a string (Section 2.6.1, Chapter 3), the letter A indicates one basic character rather than one basic symbol is to be transmitted. This is the same as S-format, except the ALGOL equivalent of the basic character is of type integer rather than a string.

Again because of the difference in string definition, the transfer function EQUIV(S) is an integer procedure whose value is the internal representation (Appendix A) of the first basic character in the string S. Thus, it has the same value as if the string S were input in alpha format.

6.2.3.3 Nonformat

The M code added to the nonformat codes indicates that the value of a single variable of any type is to be input or output in the exact form it appears on the external medium or in memory with no conversion.

All four nonformat codes I, R, L, and M input or output 16 consecutive octal digits.

6.2.3.4 Boolean Format

On input, incorrect forms cause error messages and the object program terminates abnormally.

6.2.3.7 Standard Format

The standard format for output is +D.9D +3D for real values and +15ZD for integer values. When the given format is incorrect, the modified output standard formats are '*'+D.9D+3D'*' for real values and '*'+16D'*' for integer values (Sections 6.1.3.3 and 6.1.3.7).

The number of blank characters, k, serving as a delimiter between numbers in standard format may be specified on the channel card (Chapter 5); if not, 2 is assumed.

String parameters can be output under STANDARD format, nS, where n is the length of the string.

6.3.3 Semantics
The infinite repetition of the parenthesized quantity is defined as meaning 32,767 repetitions.

7. Standard Input-Output Procedures

7.1 General Characteristics

In the input-output procedures, the term CHANNEL indicates an integer variable called by value. This value is the channel number. Channel numbers are associated with a set of characteristics on a channel card (Chapter 5). These characteristics include the SCOPE logical unit number of a hardware device, recording mode and density; physical recording characteristics, such as record size and paging factor, and so on.

Each channel is associated with a format area, which is the memory image of the external line. The area is as long as the maximum record size defined by the P parameter on the channel card. Channels used for GET ARRAY and PUT ARRAY are not associated with a format area since these procedures do not involve formatting.

All characteristics associated with a channel, including the format area, are contained in a stack of information which is retained during execution of the object program (Appendix C).

7.2 Horizontal and Vertical Control

The initial value of P on the channel card (Chapter 5) defines the maximum size of the physical record to be read or written. P may be changed during program execution, but may never exceed its initial setting. At any one time, the actual size read or written is always the current value of the P parameter. The initial value of P' on the channel card defines the number of lines per page; the value of this parameter may be changed to exceed its initial setting.

7.3 Layout Procedures

If any of the procedures FORMAT, H END, V END, H LIM, V LIM, TABULATION, or NO DATA are called when neither IN LIST nor OUT LIST is active, they have the effect of a dummy procedure; a procedure call is made and the procedure is exited immediately.

7.3.1 Format Procedures
The single procedure with call

FORMAT (STRING, X_1,X_2,...X_n)

replaces the n+1 procedures with call

FORMAT n (STRING, X_1,X_2,...X_n) n=0,1,2,3,4,5,6,7,8,9

defined in the proposal. The number of X_i variables included in the call to FORMAT defines which of the n+1 procedures (defined in the proposal) it is equivalent to. For example,

FORMAT (STRING, X_1,X_2) is equivalent to

FORMAT 2 (STRING, X_1,X_2) defined in the proposal.

A call to FORMAT may include 0-30 variables (unlike the proposal which is 0-9). The maximum number depends on the parenthesized structure of the string.

7.3.2 Limits
Since the first character of each record is used by the system to control skipping when paging is requested, the H LIM procedure increases the values of the L and R parameters by 1, to overcome the loss of this character.

7.3.5 End of Data
End of data is defined as the occurrence of an end-of-file mark (7.8 punch in the first character position of a record) on the input device.

If the procedure NO DATA is not used, transfer occurs to the label established for the channel by the EOF procedure. If the EOF procedure has not been called, the object program terminates abnormally with the message UNCHECKED EOF.

7.5.1 Symbol Transmission
Because of the definition of a string (Section 2.6.1, Chapter 3), the procedures IN SYMBOL and OUT SYMBOL are replaced by the analogous procedures IN CHARACTER and OUT CHARACTER with the calls:

IN CHARACTER (CHANNEL, STRING, DESTINATION)

OUT CHARACTER (CHANNEL, STRING, SOURCE)

IN CHARACTER examines the next basic character on the channel; if it has a value 12_8, the integer variable DESTINATION is set to -1. If not, the character is compared for equality with the characters that comprise the

string; if a match is found at the Jth character, DESTINATION is set to the value J; if no match is found, DESTINATION is set to 0.

OUT CHARACTER examines the value of SOURCE; if it is negative, the character 12_8 is output. If the value is in the range of 1 to J where J is the length of the string, the corresponding character of the string is output; otherwise, an object program error results.

In both IN CHARACTER and OUT CHARACTER, embedded string quotes ('and') are each counted as two characters, as in the procedure CHLENGTH.

7.5.2 Transmission of Type real

The procedures IN REAL and OUT REAL handle numbers in standard format.

7.5.4.1 Output

The single procedure with call:

OUTPUT (CHANNEL, FORMAT STRING, $X_1, X_2, \ldots X_n$)

replaces the n+1 procedures with call:

OUTPUT n (CHANNEL, FORMAT STRING, $X_1, X_2, \ldots X_n$)
n=0,1,2,3,4,5,6,7,8,9

defined in the proposal. The number of X_i variables included in the call to OUTPUT defines which of the n+1 procedures (defined in the proposal) it is equivalent to. For example,

OUTPUT (CHANNEL, FORMAT STRING, X_1) is equivalent to

OUTPUT 1 (CHANNEL, FORMAT STRING, X_1) defined in the proposal.

A call to OUTPUT may include 0-61 variables (unlike the proposal which is 0-9).

In Step 6 (Formatting the Output), the number of characters, s, does not depend on the output value using "A" or "S" format since these involve basic characters (s=1) not basic symbols.

In Step 9 (Finish the Item), Process D (New Line), skipping the output medium to a new line involves writing a record on the external device. The size of the record is the smallest number of whole words that can contain P characters. The successful completion of the write operation is ensured before Process D is exited.

In Step 9 (Finish the Item), Process E (New Page), skipping the output medium to a new page involves setting character 1 of the next line to a value which has significance only to the printer driver and causes page eject. On

a normal line, character 1 is set to a value which results in single spacing. This character does not appear if the external device is a printer; on any other device, it is simply the first character on the external medium.

When the user specifies paging, therefore, character 1 is not available for use, regardless of the external device. To overcome the loss of this character position, the procedure H LIM (Section 7.3.2) increases the values of the L and R parameters by 1.

If no paging is specified, the user may reference character 1; H LIM does not adjust the L and R parameter values. However, if the external device is a printer, character 1 of each record is used by the driver as described above; to avoid loss of a significant character and random page and line skipping, the user should set this character accordingly.

Process F (Page Alignment); when Process D is executed, each line skipped consists solely of blank characters.

7.5.4.2 Input

The single procedure with call:

INPUT (CHANNEL, FORMAT STRING, $X_1, X_2, \ldots X_n$)

replaces the n+1 procedures with call:

INPUT n (CHANNEL, FORMAT STRING, $X_1, X_2, \ldots X_n$)

n=0,1,2,3,4,5,6,7,8,9

defined in the proposal. The number of X_i variables included in the call to INPUT defines which of the n+1 procedures (defined in the proposal) it is equivalent to. For example,

INPUT (CHANNEL, FORMAT STRING, X_1, X_2, X_3) is equivalent to

INPUT 3 (CHANNEL, FORMAT STRING, X_1, X_2, X_3) defined in the proposal. A call to INPUT may include 0-61 variables (unlike the proposal which is 0-9).

In Step 6 (Formatting for Input), no special test is made to see if the format item is "A" to determine the length s, since this format involves a basic character (s=1) not a basic symbol.

In Step 8 (Processing Overflow), because of the definition of "A" format, only one basic character is input.

In Step 9 (Finish the Item), the mention of "A" format does not apply because of its definition; and the first sentence should read: "If any format other than "N" is being used, input s characters. Determine the value of the item that was input here, or in steps 7 and 8 in the case of "N" format, using the rules of format."

Process D (New Line), skipping the input medium to a new line involves reading a record from the input device. The size read is the smallest number of whole words that can contain P characters. The successful completion of the read operation is ensured before Process D is exited.

Process E (New Page), skipping the input medium to a new page involves the assumption that the next physical record on the input device begins the new page (control character in position 1 which is not accessible by the program, as specified in the corresponding process of output in Section 7.5.4.1).

7.5.5 Intermediate Data Storage

The procedures GET and PUT are not implemented; they have been replaced by GET ARRAY and PUT ARRAY, though these are in no way analogous. The calls are:

GET ARRAY (CHANNEL, DESTINATION)

PUT ARRAY (CHANNEL, SOURCE)

DESTINATION and SOURCE are both the names of arrays. These procedures can be used only on non-formatted channels defined on channel cards by the special character A.

GET ARRAY reads one physical record, equal in length to DESTINATION, from the channel directly into DESTINATION. The record is not stored first in a format area and no regard is made for maximum record size or paging. The record should contain the array arranged by rows, since this is how arrays are assumed to be stored in the system.

PUT ARRAY writes one physical record, equal in length to SOURCE, directly from SOURCE to the channel. The record is not stored first in a format area and no regard is made for maximum record size or paging. The physical record reflects exactly how the array is stored in memory, by rows.

7.5.7 Additional Primitives

The procedure LENGTH is replaced by its analog CHLENGTH to correspond to the substitution of IN CHARACTER and OUT CHARACTER for IN SYMBOL and OUT SYMBOL. The procedures NAME and TYPE are not implemented.

7.5.7.1 String Handling
CHLENGTH is the analog procedure of LENGTH; the difference results from the definition of a string (Section 2.6.1, Chapter 3).

CHLENGTH (STRING)

CHLENGTH is an integer procedure whose value is the length of the string in characters. Each embedded string quote (' and ') counts as two characters.

Because of the definition of a string (Section 2.6.1, Chapter 3), the procedure STRING ELEMENT assigns to the integer variable X an integer corresponding to the Ith character of the string S1 as encoded by the string S2. Effectively, an OUT CHARACTER (Section 7.5.1) process is performed on the string S1, according to the integer variable I. An IN CHARACTER process is then performed with the resultant character on the string S2, producing an integer value to be stored in the integer variable X.

7.5.7.2 Name Association
The NAME function is not implemented.

7.5.7.3 Type Determination
The TYPE function is not implemented.

HARDWARE FUNCTION PROCEDURES

In the following description of specific hardware functions, wherever the condition of an external device is mentioned, it refers only to that condition as recognized on the associated channel.

A channel is considered to be input if last used for a read operation, output if last used for a write operation, and closed if not previously referenced or if referenced by a closing procedure such as ENDFILE.

PARITY (CHANNEL, LABEL)
Each of the four procedures MANINT, ARTHOFLW, PARITY, and EOF establishes a label, LABEL, to which control transfers in the event of a manual interrupt, arithmetic overflow, uncorrectable parity error, or end-of-file condition. Each procedure can be called as many times as necessary in the course of the program to modify the label. The PARITY and EOF procedures must be called once for each channel for which a label is to be established. If a procedure has not been called or if the label is no longer accessible when the corresponding condition occurs, the object program terminates abnormally with an appropriate error message.

MANINT (LABEL)

ARTHOFLW (LABEL)

PARITY (CHANNEL, LABEL)

EOF (CHANNEL, LABEL)
If IN LIST is in operation a label may be established by the NO DATA procedure (Section 7.3.5) instead of by the EOF procedure. During the execution of the IN LIST procedure, any label established by the NO DATA procedure takes precedence over an EOF label.

MODE (CHANNEL, TYPE)
This procedure sets density or parity for the subsequent reading or writing of the external device. Density and parity are initialized on a channel card and depend on the value of TYPE, as follows:

0 No density or parity selection required
1 Do not change density, set parity to odd (binary)
2 Do not change density, set parity to even (BCD)
3 No density selection required, do not change parity
4 Set density to low (200 bpi), do not change parity
5 Set density to medium (556 bpi), do not change parity
6 Set density to high (800 bpi), do not change parity

A channel is considered to be input if last used for a read operation, output if last used for a write operation, and closed if not previously referenced or if referenced by a closing procedure such as ENDFILE.

If any of the following procedures are called for an external device which cannot perform the operation, the procedure behaves like a dummy procedure; and at the completion of the procedure the channel is considered to be closed.

SKIPF (CHANNEL)
This procedure spaces the external device forward past one end-of-file mark. It is treated as a dummy procedure on an output channel.

SKIPB (CHANNEL)
This procedure spaces the external device backwards past one end-of-file mark. On an output channel before the spacing occurs, information in the format area is written out, an end-of-file mark written, and backspaced over.

ENDFILE (CHANNEL)
This procedure writes an end-of-file mark on the external device. It is treated as a dummy procedure on an input channel. Before the end-of-file mark is written, information in the format area is written out.

REWIND (CHANNEL)
This procedure rewinds the external device to load point. On output, before the rewind occurs, information in the format area is written out, and an end-of-file mark is written and backspaced over.

UNLOAD (CHANNEL)
This procedure unloads the external device. On output, before the unloading occurs, information in the format area is written out, and an end-of-file mark is written and backspaced over.

BACKSPACE (CHANNEL)
This procedure backspaces the external device past one physical record. On output, before the backspace occurs, an end-of-file mark is written and backspaced over.

IOLTH (CHANNEL)
This procedure can be used only on non-formatted channels (those used for GET ARRAY and PUT ARRAY). It yields the number of array elements in the last read or write operation on the external device (the number in the last GET ARRAY or PUT ARRAY operation).

I/O ERRORS

At object time, two types of errors not directly concerned with programming are detected: illegal input-output operation requests and invalid transmission of data.

Illegal Input-Output Operations
If the user requests an input operation on a channel associated with a device which cannot read, or if the last operation on the channel was neither an input operation nor a closing operation (such as REWIND), the object program terminates abnormally with the diagnostic ILLEGAL IN-OUT. The same result occurs if the user requests an output operation on a channel associated with a device which cannot write, or if the last operation was neither an output operation nor a closing operation (such as ENDFILE).

Input-Output Transmission Errors

If a parity error occurs on an input tape, the tape is backspaced over the error record, and the record is re-read. If the error persists, the cycle may be repeated up to 10 times before it is considered uncorrectable.

If a parity error occurs on an output tape, the tape is backspaced over the error record, the tape spaced and erased forward 6 inches, and the error record re-written. If the error persists, the cycle may be repeated up to 10 times before it is considered uncorrectable.

On an uncorrectable parity error, control transfers to the label established for the channel by the PARITY procedure. If there is no label available, the object program terminates abnormally with the diagnostic UNCHECKED PARITY.

End-of-File

When an end-of-file is encountered on an external input device, control transfers to the label established for the channel by the NO DATA procedure (if within IN LIST only) or the EOF procedure. If there is no label available, the object program terminates abnormally with the message UNCHECKED EOF. During execution of the IN LIST procedure, a label established by NO DATA takes precedence over a label established by EOF.

An end-of-file mark is a 7, 8 punch in the first character position of a record.

End-of-Tape

If an end-of-tape is detected during writing, the tape is backspaced over the record, two end-of-file marks are written, and it is unloaded. A message, RE-LOAD LUN XX - PRESS GO WHEN READY appears on the CTO; the record is written as the first record of the newly mounted tape.

CHANNEL CARDS 5

All input-output statements (Chapter 4) specify a channel on which the operation is to be performed. A channel is referenced by a channel number and every channel is associated with a set of characteristics defined on channel cards.

Channel cards appear as the first or only cards of the object-time data on the standard input device; they are interpreted by the controlling routine before the object program is entered and are printed on the standard output device.

CHANNEL DEFINE CARD

Each channel define card describes the characteristics to be associated with one channel number. The general format of this card is:

CHANNEL, CN=LUXX, Pr, PPs, Kb, Dd, B, A

The eight characters CHANNEL, appear in columns 1-8, and each parameter describes a different characteristic. Parameters are separated by commas; blank columns may appear anywhere. The last parameter has no delimiter; but the information for one channel must be wholly contained on a single card. Only the first parameter is required; the others are optional and may be specified in any order.

CN is an unsigned integer, maximum 14 decimal digits. XX is a SCOPE logical unit number which must be declared to the SCOPE monitor with an EQUIP card (Chapter 8). This parameter indicates that whenever the number CN is used in the call of an input-output procedure logical unit XX is to be referenced.

The P parameter indicates the maximum width (r characters) of the physical page (the record) for that channel. This is equivalent to establishing a formatting area of r characters, 20 or more. When parameter P is omitted, 136 is assumed.

The PP parameter indicates the maximum length of the physical page (s records) for the channel. If PP0 is specified or if the parameter is omitted, the page length is governed by the external device.

If the user defines page width or page length beyond the capabilities of the corresponding external device, data is lost.

The K parameter determines the number of consecutive blanks that serves as a delimiter for a number read or written in standard format (Chapter 4) on the channel. The omission of this parameter is equivalent to K2.

The D parameter is meaningful only for magnetic tape. D2 sets the density to 200 bpi, D5 to 556 bpi, D8 to 800 bpi, and when D0 is used or the phrase is omitted the density is dependent on operator or installation control or selected by the MODE procedure.

The B parameter indicates reading or writing in binary (odd) parity. The absence of this parameter sets BCD (even) parity.

The A parameter is included when a channel is to be used only by the procedures GET ARRAY and PUT ARRAY which do not involve formatting of data. All page width, page length, and delimiter blank length indications are ignored if this parameter is included.

CHANNEL EQUATE CARD

Channel equate cards permit the user to associate more than one channel number with an identical set of characteristics. The card format is:

CHANNEL, $CN_1 = CN_2$

CN_1 and CN_2 are each unsigned integers with a maximum of 14 decimal digits.

CN_2 must appear on a channel define card elsewhere (though not necessarily earlier) in the set of channel cards. The characteristics defined on that card, including the same format area, can be referenced by the number CN_1 as well as CN_2. Any number of channel numbers may be equated in this way with the same channel.

CHANNEL END CARD

The last card of every set of channel cards must be in the format:

```
CHANNEL, END
```

This card indicating the end of channel information must be included even when there are no other channel cards in the deck.

STANDARD ALGOL CHANNEL CARDS

Two channel cards with standard channel numbers and characteristics are automatically supplied by the ALGOL system for the SCOPE standard input and output devices, as follows:

```
CHANNEL,60=LU60,P80
CHANNEL,61=LU61,P136,PP60
```

The two standard units may be referenced by the channel numbers 60 and 61 and do not require channel cards. They are printed as part of the channel card listing as if they were specified by the user.

Logical unit numbers 60 and 61 do not require SCOPE EQUIP cards.

Duplication of Channel and Logical Unit Numbers

The same channel number may not appear in more than one channel define card in a set. Similarly, a channel number which appears on a channel define card may not be included on the left-hand side of a channel equate card, since this is equivalent to associating that number with more than one set of characteristics. For example, the following set of cards is illegal since it associates channel number 135 with logical unit number 26 and also with logical unit number 37.

```
CHANNEL,135=LU26
CHANNEL,151=LU37
CHANNEL,135=151
```

The same logical unit number may appear on any number of channel define cards; although the channels remain completely independent of each other, all input-output operations specifying any of the different channel numbers make reference to the same logical unit. For example, in the following cards, logical unit number 45 is referenced whenever channel number 107 or 123 is specified; it is irrelevant that channel 107 is associated with formatting, and channel 123 is used only for GET ARRAY and PUT ARRAY.

```
CHANNEL,107=LU45
CHANNEL,123=LU45,A
```

These rules apply to both user defined channels and those automatically supplied by 3100/3200/3300/3500 ALGOL.

OBJECT PROGRAM FORMATS AND EXECUTION OPTIONS 6

An object program generated by an ALGOL compilation may be in a form to be loaded and executed in the normal manner directly under the SCOPE monitor, or it may be in a form to be loaded and executed in segments under control of special compiler routines. In either form, the instruction sequence is identical.

If the segmented mode of output is requested, pass 4 of the compiler prepares a segment tape for later execution (Appendix B). In effect, pass 4 modifies the program in normal form to make it acceptable to pass 5, then writes it in 512-word segments on the segment tape.

It is possible to compile a main program or a procedure in either mode. In addition, a segment tape can be produced from only a load-and-go tape.

NORMAL MODE

In this mode, the binary form of the program conforms completely to the specifications defined in the SCOPE/COMPASS Reference Manual.

All object programs in this form are assigned the entry point ALGOLRUN, the entry point to the library routine which controls execution of the object program. The ALGOLRUN routine is loaded by the SCOPE loader during normal processing. All object programs in this form have a DATA allocation of 192 (300_8) words and a COMMON allocation of 0 words.

It is possible to compile a main program and any number of pre-compiled procedures to the load-and-go tape separately, in any order, within the same job; so that this load-and-go tape can be executed as an entity or converted into a segment tape for execution in segmented mode.

If the user executes the main program in conjunction with non-ALGOL subprograms he must ensure that:

The subprogram with the largest DATA allocation is loaded first (this is a loader requirement). It may be the main program.

The first 192 (300_8) words of DATA may not be utilized in any of the other subprograms.

SEGMENTED MODE

In the segmented form, the program is structured in 512-word segments, as described in Appendix B, each of which appears as one record on the segment tape. The segment tape may be executed directly following its preparation as part of the same compilation, or it can be executed in a completely separate process.

Execution of the segment tape is controlled by pass 5 of the compiler. This routine keeps a record of each segment currently in memory. When a segment is referenced, pass 5 determines if it is already in memory; if it is not, pass 5 loads it into available memory. Segments are loaded and retained until available memory is filled and space is required by another segment or by the object program stack (Appendix C). Object program execution requires space for at least two segments, otherwise execution cannot begin or continue normally.

Segments are constructed so that they may be overlaid when no more memory is available; and if required again later, read back in from the segment tape any number of times.

The segmented mode of execution must be requested when the object program and its data requirements will not fit as a whole into available memory. It may also be used for smaller programs to avoid returning control to SCOPE for loading the object program.

The maximum size of an object program produced in a single compilation is 32K (32,768) words. The maximum size of a program on the segment tape, however, can be 512 segments (262,144 words), produced by the combination of a main program and up to 50 pre-compiled procedures.

ALGOL CONTROL CARD

The ALGOL control card specifies the input and output options which the user requires. This card also causes the SCOPE monitor to call the ALGOL compiler from the library (SCOPE/COMPASS Reference Manual). The format of the card is free-field; the parameters may appear in any order, separated by commas, and may contain any number of blanks. The general format of the card is:

$^{7}_{9}$ALGOL,I,A,L,X,P,S,G,R,C,B,D,N

All options, excluding C and N, can be followed by =n, where n defines a logical unit number. When =n is omitted, the standard logical unit is used for the corresponding option. Each option must begin with the letter shown, though additional letters may follow it and are ignored. For example, L and LIST are equally acceptable for the list parameter.

I	Specifies the logical unit number for the source input (standard unit is 60)
A	List the assembly language form of the object program (standard unit is 61)
L	List the source language (standard unit is 61)
X	Write the object program in binary relocatable non-segmented form on the load-and-go unit (standard unit is 56)
P	Punch the object program in binary form (standard unit is 62)
S	Produce the segment tape (standard unit is 55)
G	Use only the binary decks of the load-and-go unit to prepare the segment tape (standard unit is 56)
R	Execute the segmented program on the segment tape (standard unit is 55)
C=m	Compile a procedure, rather than a program, from the input device. The procedure is associated with the code number m, 0-99999. If fewer than five digits are specified, the number is zero-filled on the left (20 is equivalent to 00020)
B	Punch the assembly language form of the object program (standard unit is 62)

D=n Prepare the special tape which will be used in the case of abnormal object program termination to cross-reference the dump of the declared variables with the corresponding identifiers. (No standard for this option; the logical unit number, n, must always be specified.) See Appendix D for the object-time dump formats.

N Do not generate the array bounds checking code in the object program (Section 3.1.4.2, Chapter 3).

Except for the I option, the absence of any parameters indicates the corresponding option is not required. If I is absent, the standard input device (logical unit number 60) is used for the source deck. (Since I with no =n specification and the absence of I have the same meaning, I should be used only with =n to indicate a unit other than the standard one for the source deck.)

Any illegal, contradictory, or meaningless combinations of parameters are diagnosed by the compiler, which then makes a legal selection from the set specified and continues compilation. The compiler diagnostic message is:

ERROR IN CONTROL-CARD, OPTION xx IS DELETED

OBJECT PROGRAM OUTPUT OPTIONS

The user may request the object program in normal binary relocatable non-segmented form on either the punch or the load-and-go unit or in the segmented form on the segment tape, as below. The format of the punched program is identical to the load-and-go format.

Create Load-and-Go Tape (Option X)

A load-and-go tape can be created in two ways: from a main program on the input device and from a procedure on the input device. In both cases, the load-and-go tape is created on logical unit 56, unless the user requests otherwise on the control card.

The format of the load-and-go tape conforms to the SCOPE specifications; that is, the binary deck is followed by an end-of-file mark and the tape is positioned after the deck and ahead of the mark.

The load-and-go tape may be executed normally under control of SCOPE with SCOPE LOAD and RUN cards. The SCOPE loader loads all binary decks between load point and the first end-of-file mark on the tape (SCOPE/COMPASS Reference Manual).

Produce a Segment Tape (Option S)

There are three ways in which a segment tape can be produced: from a main program on the input device, from a procedure on the input device, and from binary decks on a load-and-go device. The segment tape is produced on logical unit number 55, unless the user requests otherwise on the control card.

Main Program on Input Device

In this mode, the main program on the input device is compiled into a segmented program on the segment tape. All binary decks residing on the load-and-go tape are also incorporated into this segment tape.

The binary decks should be procedures referenced by the main program (or each other) and compiled previously by the ALGOL compiler in load-and-go mode.

If the compiler detects the presence of a main program already on the load-and-go unit, it terminates compilation and returns control to the SCOPE monitor.

Procedure on Input Device

Operation is the same as for a program on the input device except that it is the absence of a main program from the load-and-go unit that causes the compiler to terminate the job and return control to the SCOPE monitor.

Binary Decks on Load-and-Go Unit (Options G and S)

In this mode, only the binary decks on a specified load-and-go unit are included in the segmented program on the segment tape.

These binary decks should consist of only one main program and up to 50 pre-compiled procedures. If the compiler detects none or more than one main program on the load-and-go unit, it terminates the job and returns control to the SCOPE monitor.

In all three cases, if the main program or any of the pre-compiled procedures reference other procedures not on the load-and-go unit, the compiler searches the library unit and incorporates them into the segment tape.

Any standard procedures (such as SIN, COS) referenced by the main program or pre-compiled procedures, are also incorporated from the library into the segment tape.

A segment tape may be executed within the same compilation process or may be executed later in a completely separate process. If executed in the same process, any unit declared for the R option (meaning execute) must be the same as that declared for the S option. If omitted, logical unit number 55 is assumed to contain the segmented program.

If a segment tape is executed separately from the process in which it is prepared, the resident monitor must be identical at compile time and execution time since the relocation already performed on the program during the segmentation process takes into account the memory situation at compile time. Logical unit number 55 is assumed to contain the segment tape, unless the user declares otherwise on the control card.

EQUIPMENT REQUIREMENTS 8

Compilation requires a minimum of five input-output units:

1 input unit

1 output unit

1 library unit

2 scratch units

The input unit can be a card reader or a tape unit; the output unit can be a printer or a tape unit.

SCRATCH UNITS

The compiler selects for its two scratch tapes the units declared for the segment tape and the load-and-go tape. When the user does not specify either on the ALGOL control card, the compiler automatically uses logical unit numbers 55 and 56.

During compilation of a source deck, each pass stores intermediate information in available memory until it is full or until all information has been processed. If the information fits in available memory, it is retained there for the next pass. Otherwise, the compiler writes out the intermediate information in records on the scratch tapes as it is processed. This information is used as input to a subsequent pass of the compiler. To avoid rewinding the scratch tapes, the intermediate information is read back in reverse order from which it is written. If the scratch unit does not possess the read backward facility, the compiler simulates the action by one backspace, one read forward, and one backspace operation.

At the beginning of compilation, the compiler writes an end-of-file mark on each of the scratch tapes exactly at the current positions, and the tapes are used beyond this end-of-file mark only.

LOAD-AND-GO UNIT

If the ALGOL compilation calls for the preparation of a segment tape, all binary decks on standard load-and-go between load point and the position of the tape at the beginning of compilation are incorporated into the segmented program.

Since, however, the standard load-and-go unit (logical unit 56) is a SCOPE system unit, it is always rewound to load point at the beginning of each job. Therefore, the ALGOL compiler will not find any decks to include in the segment tape unless they are written during the same job and prior to the ALGOL compilation making the segment tape.

The load-and-go tape is assumed to conform to SCOPE specification; that is, the last binary deck written on the tape is followed by an end-of-file mark and the tape is positioned following the deck but ahead of the mark, ready for further writing or processing.

For example, a load-and-go tape can be written as output from an ALGOL compilation within the same job as a segmented mode ALGOL compilation; or the tape can be written as a result of an XFER of a binary deck punched during a completely separate ALGOL compilation.

At the end of compilation, the load-and-go unit is positioned exactly where it was at the beginning of compilation, unless the compilation called for load-and-go output. In this case, the tape conforms with SCOPE specifications; that is, the binary deck written is followed by an end-of-file mark and the tape is positioned following the deck and ahead of the mark.

When a segment tape is being made directly from a load-and-go tape, with no source input, the load-and-go unit need not be the standard one nor need it have been written during the same job, since the process does not really involve the use of the tape as a scratch tape. The tape must, however, conform to SCOPE specifications for a load-and-go tape (SCOPE/COMPASS Reference Manual).

EQUIPMENT DECLARATIONS

A SCOPE equipment declaration is not necessary if the user does not explicitly request units other than the standard ALGOL scratch units (logical units 55 and 56). However, if he specifies non-standard units, he must include a SCOPE EQUIP card for each unit in his job deck. If only the SCOPE standard units are to be used for object program execution, no EQUIP card is necessary; otherwise EQUIP cards must be included.

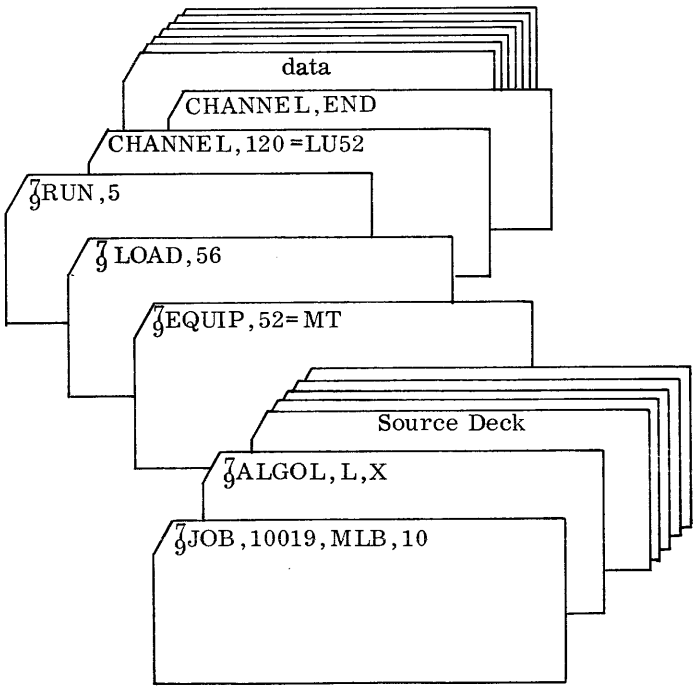
Except for a segment and run compilation, compilation and object-time equipment declarations appear separately in the deck. Compilation EQUIP cards must appear just before the ALGOL control card; object-time EQUIP cards must appear just prior to the LOAD card for load-and-go execution.

In a segment and run compilation, the object program EQUIP requirements must be declared at the same time as compilation requirements (before the ALGOL control card) since there is no logical break between compilation and execution and no return to SCOPE. There can be no overlapping of the equipment declarations for compilation and execution.

To create a logical break in the segment and run process, the user can specify two ALGOL compilations: the first to prepare the segment tape; the second to execute it. Moreover, he can now include separate EQUIP card declarations for each part of the process.

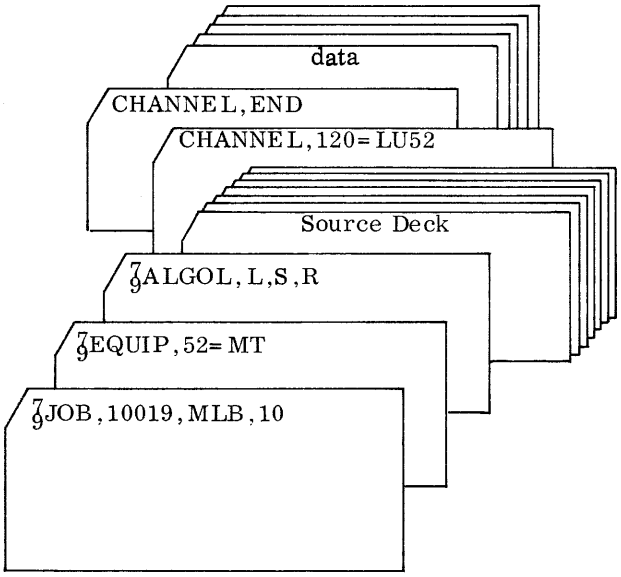
TYPICAL DECK STRUCTURES

Compile a load-and-go tape and execute



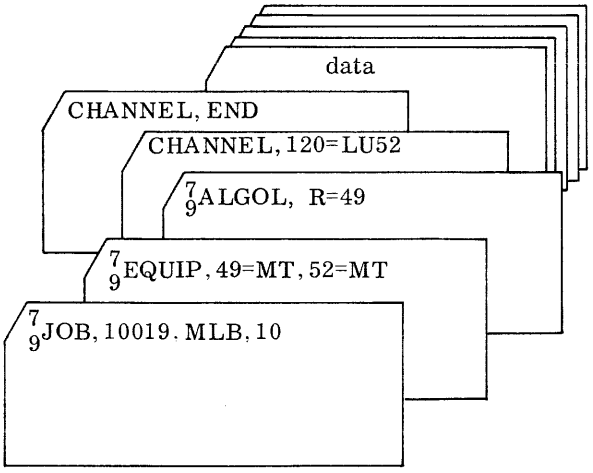
Logical unit 52 is assigned to ALGOL channel 120

Compile Program to Segment Tape and Execute



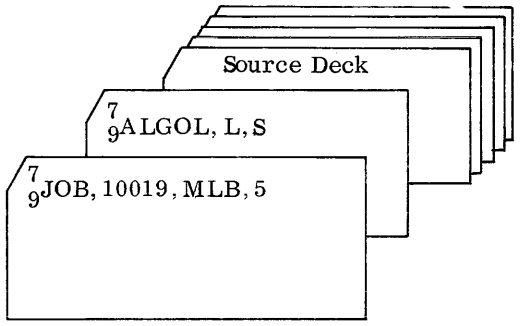
Logical unit 52 is assigned to ALGOL channel 120 for use at object time.

Execute a Segment Tape

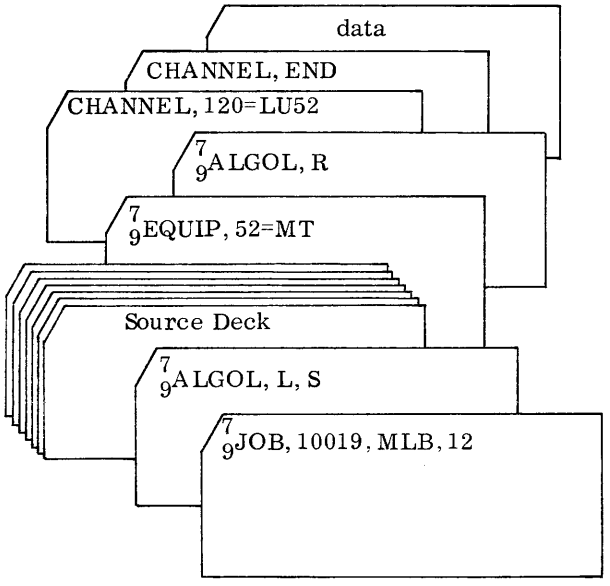


Segment tape created previously and assigned to logical unit 49 for this job. Logical unit 52 is assigned to ALGOL channel 120.

Compile Program to Segment Tape

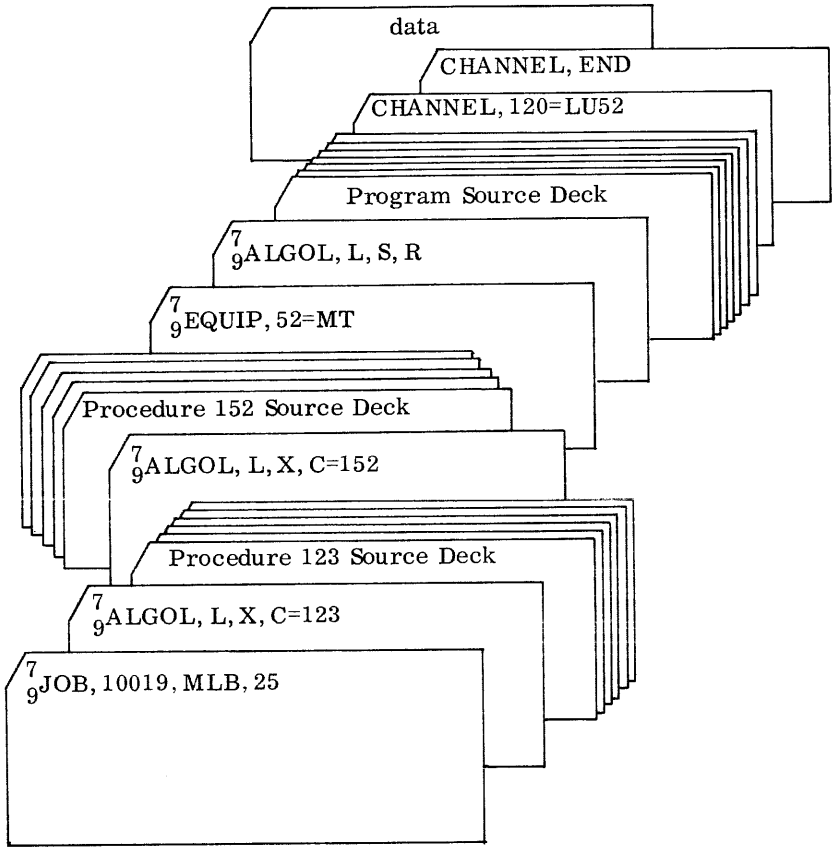


Compile Program to Segment Tape; Return to SCOPE, then Execute



Logical unit 52 is assigned to ALGOL channel 120.

Pre-Compile two Procedures onto Load-and-Go Tape; Compile Segment Tape from Main Program to Include these two Procedures: Execute



Procedure 123 is compiled to load-and-go tape; procedure 152 is compiled to load-and-go tape. The main program is compiled to a segment tape with procedures 123 and 152 included. Logical unit 52 is assigned to ALGOL channel 120.

APPENDIX SECTION

ALGOL CHARACTER TABLES A

The 48 characters available in ALGOL source (hardware) language are shown below. Any of the complete 64 6-bit character set may appear in data or (except for 12_8) in strings.

Table 1. ALGOL Character Set

Character	Card Punch	Internal Octal	Character	Card Punch	Internal Octal
A	12 - 1	21	Y	0 - 8	70
B	12 - 2	22	Z	0 - 9	71
C	12 - 3	23	0	0	00
D	12 - 4	24	1	1	01
E	12 - 5	25	2	2	02
F	12 - 6	26	3	3	03
G	12 - 7	27	4	4	04
H	12 - 8	30	5	5	05
I	12 - 9	31	6	6	06
J	11 - 1	41	7	7	07
K	11 - 2	42	8	8	10
L	11 - 3	43	9	9	11
M	11 - 4	44	+	12	20
N	11 - 5	45	-	11	40
O	11 - 6	46	*	11-4-8	54
P	11 - 7	47	/	0 - 1	61
Q	11 - 8	50	=	3 - 8	13
R	11 - 9	51	(	0-4-8	74
S	0 - 2	62	)	12-4-8	34
T	0 - 3	63	.	12-3-8	33
U	0 - 4	64	,	0-3-8	73
V	0 - 5	65	'	4 - 8	14
W	0 - 6	66	$	11-3-8	53
X	0 - 7	67	␣ †	␣	60

† blank column

Table 2. Character Representation of ALGOL Symbols

ALGOL Symbol	48-Character Representation	ALGOL Symbol	48-Character Representation
A - Z	A - Z	true	'TRUE'
a - z	~	false	'FALSE'
0 - 9	0 - 9	go to	'GO TO'
+	+	if	'IF'
-	-	then	'THEN'
×	*	else	'ELSE'
/	/	for	'FOR'
↑ †	'POWER'	do	'DO'
÷	'/' or 'DIV'	step	'STEP'
>	'GREATER'	until	'UNTIL'
≥	'NOT LESS'	while	'WHILE'
=	= or 'EQUAL'	comment	'COMMENT'
≠	'NOT EQUAL'	begin	'BEGIN'
≤	'NOT GREATER'	end	'END'
<	'LESS'	own	'OWN'
∧	'AND'	Boolean	'BOOLEAN'
∨	'OR'	integer	'INTEGER'
≡	'EQUIV'	real	'REAL'
¬	'NOT'	array	'ARRAY'
⊃	'IMPL'	switch	'SWITCH'
.	.	procedure	'PROCEDURE'
,	,	string	'STRING'
:	..	label	'LABEL'
;	.,	value	'VALUE'
10 ††	'	code†††	'CODE'
␣	␣	segment†††	'SEGMENT'
(	(		
:=	.= or ..=		
)	)		
[	(/		
]	/)		
‘	'('		
’	')'		

† in a string format is represented by an asterisk.
†† ALGOL symbol$_{10}$ is a subscript digit.
††† code and segment symbols are not defined in the ALGOL-60 Revised Report

OBJECT PROGRAM STRUCTURE B

The object program produced by an ALGOL compilation may be in the normal binary relocatable non-segmented form or it may be in the special segmented form.

As described in Chapter 6, the segmented form can be derived directly, in whole or part, from a program already in the normal form. Alternatively, the segmented form can be derived directly from the source input. Since the latter case can be considered as a process in which the normal form is generated first followed by the segmentation process, a segmented program can be considered in both cases to have been derived from a program which is in the normal form.

The segmentation process does not affect the instruction sequence; it adjusts the address fields in those instructions which reference other locations in the program, both in the same segment and in different segments, so that the separate segments may be freely relocated at execution time.

To maintain compatibility between the two forms, pass 4 of the compiler generates the object program with the following addressing conventions:

Program Reference in the same Segment

Normal Form

In the normal form, such addresses have a 15-bit form, XXYYY, where XX is a 6-bit segment number (0-63) and YYY is a 9-bit relative location (0-511) in a segment. The address is assigned positive program relocatability. Such an address is the normal form of a subprogram address and thus results in the correct absolute address when relocated by the SCOPE loader relative to the beginning of the subprogram.

Segmented Form

The pass 4 segmenter recognizes all instructions with positive program relocatable address fields. To create the segmented program, it modifies the 15-bit address form XXYYY to the 15-bit address form OOYYY and flags the address as requiring further relocation at execution time. When the segment is loaded, the address is changed to the first-word address of the segment (the segment relocation factor) plus OOYYY.

Program Reference in a Different Segment

Normal Form

In the normal form, such a reference consists of a call to a controlling routine within ALGOLRUN, with the destination address as a parameter to this call. This parameter is an 18-bit complement of an address XXYYY where XX is a 6-bit segment number (0-63) and YYY is a 9-bit relative location (0-511) in segment. The address is assigned negative program relocatability. When relocated by the SCOPE loader relative to the beginning of the whole subprogram, this address results in the complement of the correct absolute address.

At execution time, the controlling routine performs the instruction with the complement of this complemented address.

Segmented Form

The pass 4 segmenter recognizes all instructions with negative program relocatable address fields. (Relocatability is used as a flag for the segmenter to distinguish between reference to the same and different segments.)

To create the segmented program, the segmenter complements the address field (to the 15-bit XXYYY form) and then modifies the segment number portion XX to MM. This modification is required since the tape segments are numbered consecutively beginning with 1, whereas the segment numbers in each subprogram included on the tape all begin with the segment number 0. The segment number is therefore increased by the number assigned to the first segment of the current subprogram.

For example, in the first subprogram on the segment tape, all segment numbers are increased by 1. If this subprogram contains 3 segments (numbers 1, 2, and 3), the first segment of the next subprogram is assigned to segment 4. Thus, all segment numbers in that segment are increased by 4, and so on for subsequent segments.

At execution time, the controlling routine interprets the modified address as the first-word address of the MM segment, plus OOYYY.

External Reference to Standard Library Procedures

The object-time addresses of the standard procedures appear in a table called STANLIST. This table contains one entry for each of the standard procedures; therefore, a unique constant, which is the relative position in STANLIST of the corresponding address entry, can be associated with each standard procedure.

The standard procedures are organized on the library in 512-word subprograms each of which contains several standard routines (such as SIN, COS). The entry points of these subprograms are the entry points of the individual routines. Each subprogram contains a 192-word (300_8) DATA allocation which is explicit only in the positions corresponding to the STANLIST entries for the standard procedures in that subprogram. Each such DATA declaration consists of an instruction whose address field is the same as described for a program reference in a different segment.

Normal Form

In this form, the STANLIST table is assigned to DATA. When the SCOPE loader overlays DATA of the main program with the DATA of the library subprograms, the corresponding STANLIST entries are filled in.

An XNL card entry is generated for the name of each standard procedure referenced. Each reference generates a jump to a controlling routine followed by the constant associated with the standard procedure referenced.

At execution time, the controlling routine uses the constant to locate the proper STANLIST entry and the complement of the correct absolute address of the corresponding standard procedure.

Segmented Form

The XNL card entries in the normal form indicate that the pass 4 segmenter should include the corresponding library routines in the segment tape. As the segmenter encounters each DATA declaration in the library subprograms, it applies the modifications to the address as described above, and superimposes the resulting address on a skeleton form of STANLIST. The STANLIST table is written as the last record on the segment tape.

Before execution of the program, pass 5 reads the STANLIST record into memory to reside there permanently during program execution. When a reference to a standard procedure is executed, the controlling routine uses the constant to locate the proper entry in STANLIST; this contains an address of the form MMYYY which the controlling routine interprets as described for a program reference in a different segment.

External Reference to Pre-Compiled Procedure

Normal Form

In this form, an XNL card entry exists for each procedure declared in a program. (The XNL name is of the form CDPxxxxx, where xxxxx is the code number assigned to the procedure when compiled separately (Section 5.4.6, Chapter 3). At the point in the program where the pseudo procedure declaration is made (the code declaration), the compiler generates a jump to a controlling routine plus an instruction reference to the corresponding external entry point name. For each call to the procedure, the compiler generates a jump to the pseudo declaration.

The XNL card entry appears in the binary deck in the same segment portion as the pseudo declaration.

Segmented Form

The XNL card entries in the normal form indicate that the pass 4 segmenter should include the corresponding routines from the load-and-go unit or the library.

During the preparation of the segment tape, the segmenter adds one entry to the STANLIST skeleton for each psuedo procedure declaration. In addition, it changes the external reference generated for each call to the procedure to a constant which is the relative position in STANLIST of the entry added for the procedure. The remainder of the processing is exactly as described above for references to the standard library procedures in segmented mode.

OBJECT PROGRAM STACK C

According to the rules of the ALGOL language, a variable is active (available for reference) in any block to which it is local or global. A variable is local to the block in which it is declared and global to the sub-blocks within the block in which it is declared.

Depending on the block structure and the variables declared at each level, not all variables are active at the same time. The object programs produced by ALGOL overlay those variables which are not simultaneously active. The overlay process is described below.

During the execution of an object program, all variables are contained in a variable-length memory stack consisting of 48-bit entries, one or more pertaining to each active variable. Since the stack includes only active entries, the size fluctuates.

During compilation, the compiler assigns to each variable an address relative to the stack reference for the block in which that variable is declared, in the reverse order of their declaration. The stack reference for each block is the position in the stack where the entries for that block are assigned at object-time. It is derived as follows.

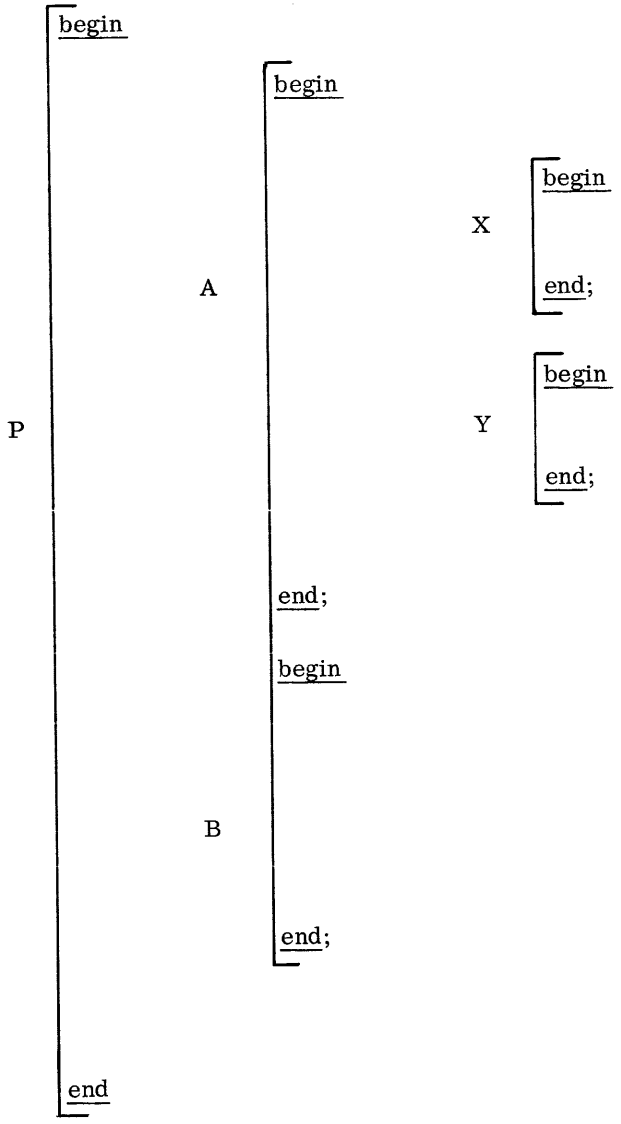
When a new block is entered which is nested in the last block entered, the stack reference for the new block is assigned to the first available (inactive) position in the stack. In addition, certain preliminary information is set into the stack, beginning at this reference point.

The compiler assigns a block level number to each block in the program, and the object program maintains 32 display entries each of which contains the stack reference for blocks at the corresponding level. The display entry corresponding to the new block is set to contain the new stack reference. Since there are 32 display entries, a program may contain a block structure in which blocks are nested up to a depth of 32 levels.

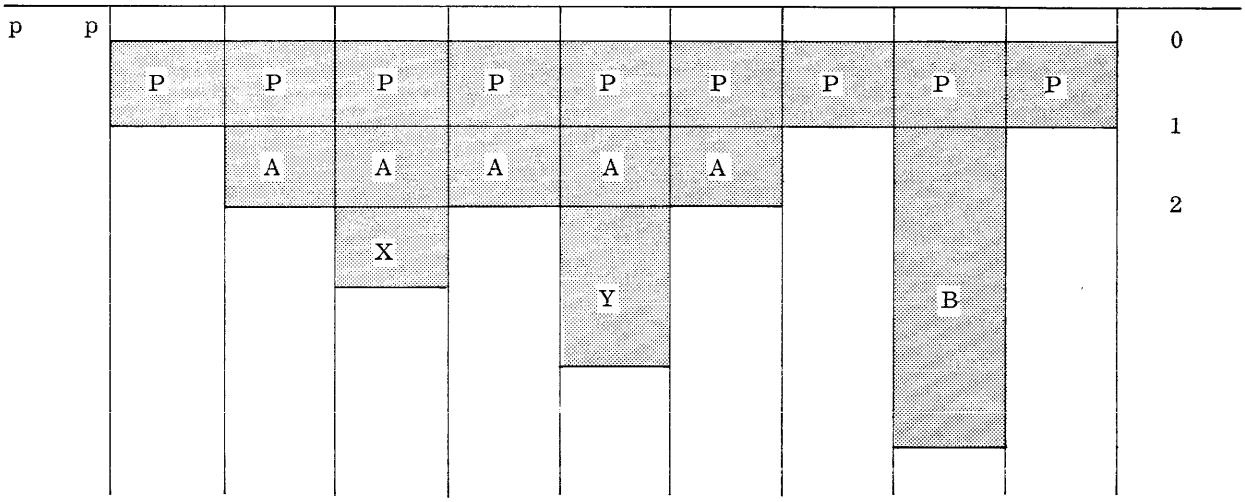
When a block is exited, the space in the stack occupied by its local variables is released as the variables become inactive. The display entry corresponding to the block being exited necessarily contains the stack reference for this block (the point up to which the stack can be released).

A goto reference from one block in a nest to an outer one results in an exit from that block and from all of the blocks up to but not including the referenced block. Thus, the effect is to change the environment of the active variables to be only those local or global to the referenced block.

When a procedure call is made, the current environment (or record of it) is preserved, since a return must be made to it at the completion of this call. The environment in which the procedure is declared is established, the procedure is executed with the corresponding variables available to it, and then the original environment is re-established. Consider the following program outline:



Block P is the program itself at level number 0; blocks A and B are at the same level (number 1) within P; blocks X and Y are at the same level (number 2) within A. The changes in the stack can be visualized as follows:



After block P is entered, the stack reference in the first display-entry is set to address p; the stack is then active up to but not including point p'.

After the nested block A is entered, the stack reference in the second display-entry is set to address p'(=a); the stack is then active up to but not including point a'.

After the nested block X is entered, the stack reference in the third display-entry is set to address a'(=x); the stack is then active up to but not including point x'. After block X is exited, the next available position in the stack is at address x (all variables declared in block X are now inactive and may be overlaid).

After block Y is entered, the stack reference in the third display-entry need not be changed; the stack is now active up to but not including point y'. After block Y is exited, the next available position in the stack is at address x (all variables declared in block Y are now inactive and may be overlaid). After block A is exited, the next available position in the stack is at address a (all variables declared in block A are now inactive and may be overlaid).

After block B is entered, the stack reference in the second display-entry need not be changed; the stack is then active up to but not including point b'. After block B is exited, the next available position in the stack is at address a (all variables declared in block B are now inactive and may be overlaid).

Stack Entries

All stack entries are 48-bits long. The different types of variables require different number of stack entries, as follows.

A simple variable requires a single stack entry containing the value of the variable.

A label or procedure requires a single stack entry containing the program location of the label or procedure and the stack reference of the environment in which the label or procedure is declared.

A switch requires one stack entry to describe the switch characteristics and one stack entry to describe each element of the switch declaration. The stack entry describing the switch contains the number of elements in the switch and the stack location of the list of stack entry elements.

An array requires one stack entry to describe the characteristics of the array, one stack entry to describe each dimension of the array, one stack entry to describe the constant which is used to check the correct overall bounds of the array, and one stack entry for each element in the array itself.

The stack entry which describes the complete array contains the stack location of the actual array elements and the stack location of the stack dimension entries.

Own Variables in the Stack

All own variables are assigned entries in the stack prior to the entries assigned to the outermost block of the program (the program itself). Thus, own variables are treated as global in definition (local to the whole program), though they are only local in scope to the block in which they are declared, just like other variables.

In particular, for an own array, the stack entries for each element in the array appear prior to the entries for the outermost block; the other entries for the array appear in the normal position in the declaration block.

Stack Listing

The object program controlling system includes a routine which produces the active contents of the stack in a meaningful format in the case of abnormal object program termination (Appendix D).

ALGOL DIAGNOSTICS D

There are four types of diagnostics associated with the ALGOL compiler system: compiler diagnostics, compiler I/O messages, object-time diagnostics, and object-time I/O messages.

COMPILER DIAGNOSTICS

Every error detected during compilation causes a diagnostic to be printed following the source listing. Each card of the source deck is assigned a line count which is printed as part of the source listing. Each compiler diagnostic includes the line count of the source card in error and a brief summary of the error condition.

Five of the compiler diagnostics are compiler messages only and have no effect on the process; PROGRAM ENDS and SOURCE DECK ENDS appear at the end of every compilation, regardless of errors.

NOTE: If the line counts of the PROGRAM ENDS and SOURCE DECK ENDS differ by a large number the programmer should make certain that part of his program has not been treated as a commentary because of a missing begin symbol or similar error.

The remaining diagnostics cause suppression of generated code, regardless of user request; some also result in the diagnostic STOP COMPILATION which terminates compilation, and any diagnostics detected after such an error condition are lost.

Message	
ARRAY BOUND TYPE	Array bound expression is not arithmetic
ARRAY BOUND - LOCAL	Variable specified for array bound is declared at same level as array
ARRAY OR SWITCH CALL	Identifier used as an array or switch has not been so declared
ARRAY, SWITCH, PROCEDURE	Too many subscripts or switch elements; or formal or actual parameters
BYPASS OVERFLOW	Capacity of compiler to handle forward references has been exceeded
CALL PARAMETER	Undeclared or untyped parameter in a procedure call
CALL PARAMETER COUNT	Procedure is called with the wrong number of parameters
CHARACTER	Illegal character (such as 12_8) found in source text

Message	
CHECK SUM	Check sum error in a binary program on load-and-go tape
'CODE' INTEGER	Literal following the symbol code is not an integer
'COMMENT'	Symbol comment in an illegal position in source text
COMPOUND DELIMITER	Hardware representation of an ALGOL symbol is incorrect (e.g., 'BIGIN')
DATA PART	DATA address in binary program on load-and-go unit not in range of STANLIST (Appendix B)
DECLARATION CAPACITY	Too many variables declared in a block structure
DECLARATION CODE O-FLOW	Capacity of the compiler to store labels, procedures, etc., for declaration code is exceeded
DELIMITER	Incorrect delimiter for the particular context appears in source text
DELIMITER IN COMMENT	Statement may have been bypassed because of missing delimiter (message only)
DELIMITER MISSING	Delimiter expected at this point in source text not found
DOUBLE DECLARATION	Identifier declared more than once in same block heading
DOUBLE SPECIFICATION	Formal parameter specified more than once in same procedure heading
'ELSE'COUNT OVERFLOW	Capacity of the compiler to handle nested if statements has been exceeded
'END'S MISSING	More begin than end symbols when 'EOP' encountered
'EOP' GEN. BY (PAR ERR) (BIN CARD) (EOF CARD)	'EOP' (end of source deck) forced at this point by parity error, binary card, or EOF card
EXTERNAL STACK OVERFLOW	Too many subprograms on load-and-go tape
'FOR' CONTROL VARIABLE	Control variable of for statement must be simple or subscripted arithmetic
FLOATED INTEGER	Integer contains more than 14 digits (message only)
FLOAT-FIX OVERFLOW	Conversion from floating-point to fixed-point exceeds 48 bits
FORMAL MISSING	Value or specification appears for an identifier not in formal list
IDC CARD	Error in IDC card in binary program on load-and-go tape: either COMMON is not 0 or DATA is not 192 (300_8)
IDENTIFIER OVERFLOW	No room in available memory to store complete list of identifiers (symbol table overflow)

Message	
'IF' CLAUSE TYPE	Expression following an if symbol must be Boolean
'IF' EXPRESSION TYPE	Expression following symbols then and else in if statement must be same type
INADMISSABLE CARD	Binary card with an inadmissable word count on load-and-go tape
INADMISSABLE RELOCATION	Incorrect relocation in binary program on load-and-go tape
INSTRUCTION UNDER-COUNT	Compiler estimate of program size incorrect
LABEL	Identifier used as a label not so declared
LOCAL VARIABLE OVERFLOW	Too many local variables defined in same block
LONG IDENTIFIER	Identifier exceeds 256 characters (message only)
MACHINE ERROR	Machine or compiler malfunction
MISSING DECLARATION	Undeclared identifier used
NO 'CODE' INTEGER	Integer expected after symbol code is missing
NUMBER SIZE	A number exceeds the floating-point capacity of the machine
NUMBER SYNTAX	A number is incorrectly punctuated
OPERAND	Incorrect operand in source text for the particular context
OPERAND MISSING	Operand expected at this point in source text not found
OPERAND OVERFLOW	Capacity of compiler to handle operands within the same statement has been exceeded
OPERATOR OVERFLOW	Capacity of compiler to handle nested operators exceeded
'OWN' BOUNDS	Bounds in an own array must be constants
PARAMETER COMMENT	Parameter comment which replaces a comma in a procedure declaration or procedure call is incorrectly formed
PRE-COMPILATION 'OWN'	Pre-compiled procedures may not contain own variables
PROCEDURE IDENTIFIER	Identifier in procedure call is not declared as a procedure
PROGRAM ENDS	This message indicates line on which the program ends
REDECLARATION CAPACITY	Capacity of compiler to handle similarly spelled identifiers in a nested block structure has been exceeded
SECOND DECLARATION	Indicates line on which second element of DOUBLE DECLARATION is made
'SEGMENT'	Symbol segment allowed only within comment
SEQUENCE	Binary cards on load-and-go tape are out of sequence

Message	
SIMPLE 'FOR' ELEMENT	Expected arithmetic expression in a for statement not arithmetic
SOURCE DECK ENDS	This message indicates line on which 'EOP' is found or forced
SPECIFICATION MISSING	Specification is missing for identifier included as a formal
STANDARD FUNCTION PARAM	Parameter in a call to a standard procedure of incorrect type
'STEP' ELEMENT TYPE	The third expression in a step element must be arithmetic
STOP COMPILATION	Indicates compilation stops at this line; error messages for other lines may be lost. Appears in conjunction with OPERAND OVERFLOW, MACHINE ERROR, etc.
STRING	Too many characters in string
STRING CHARACTER	Illegal character in a string (e.g., 12_8)
STRUCTURE CAPACITY	Compiler capacity to handle a nested structure, such as parenthetical statements, exceeded
SUBPROGRAM MISCOUNT	Incorrect number of subprograms on load-and-go or library tape
SUBPROGRAM SIZE	Size of current subprogram has exceeded 32K words
SUBSCRIPT COUNT	Array or switch called with incorrect number of subscripts
SUBSCRIPT TYPE	All subscript expressions must be arithmetic
'SWITCH' PARAMETER	All elements in a switch list must be labels or designational expressions
TERMINATION	Language construction in source text terminates illegally
TOO MANY 'BEGINS'	A block structure contains blocks nested to more than 32 levels
TOO MANY IDENTIFIERS	Too many differently spelled identifiers in the program
TOO MANY WORKING LOCS	Too many working locations in excess of declared variables are required to perform operations specified in this block
TYPE	In a general expression, elements specified must have same types
VALUE SPECIFICATION	Value applied to formal parameter whose specification does not permit a value (e.g., a label)
'WHILE' ELEMENT TYPE	Second expression in a while statement must be Boolean
XNL CARD	Error in XNL card in binary program on load-and-go tape

COMPILER I/O MESSAGE ON STANDARD OUTPUT UNIT

ALGOL-I/O-ERROR yy ON LU xx

Uncorrectable error occurred during loading one pass of compiler from the library or during compilation. The type of error is indicated by yy as follows:

yy =	PA	Irrecoverable parity error
	BC	Inadmissible binary card or relocation error
	CS	Check sum error
	LD	Lost data
	ET	EOT

The compiler will automatically terminate the compilation and return control to the SCOPE monitor.

COMPILER I/O MESSAGE on CTO UNIT

RE-LOAD LUN xx - PRESS GO WHEN READY

xx indicates device:

The card hopper is empty but not all cards to complete an ALGOL compilation have been read. Refill hopper. Press GO to continue.

The card punch has failed to feed, or a card has been mispunched. When corrected, press GO to continue.

The end of tape has been encountered on a compiler output tape; the tape has been backspaced, two end-of-file marks written on it, and unloaded. Mount a new tape on the designated unit. Press GO to continue.

Re-load printer with paper. Press GO to continue compilation.

OBJECT-TIME DIAGNOSTICS

Upon normal exit from an object program, the contents of all non-empty output format areas (Chapter 4) are output; the following message printed on the standard output device indicates a successful execution.

END OF ALGOL RUN

Upon abnormal termination of an object program, a diagnostic is printed on the standard output unit to indicate the nature of the error. The contents of the non-empty output format areas are output as above. Information which traces the execution path through the blocks in the currently active block structure is then printed on the standard output unit as follows:

```
THIS ERROR OCCURRED AFTER LINE XXXX
IN THE BLOCK ENTERED AT LINE XXXX
  (global stack information)
  (local stack information)
THIS BLOCK WAS CALLED FROM LINE XXXX
IN THE BLOCK ENTERED AT LINE XXXX
  (local stack information)
THIS BLOCK WAS CALLED FROM LINE XXXX
```

The stack information (Appendix C) for each block is printed following the corresponding BLOCK ENTERED line. If the user requests the dump tape (control card option D), this information is formatted so that each variable is associated with its source text identifier. Otherwise, the stack is printed with no special formatting.

Message	
ALPHA FORMAT ERROR	Output value is too large
ARITHMETIC OVERFLOW	Evaluation of an expression results in arithmetic overflow (e.g., division by zero) for which no provision has been made with ARTHOFLW procedure
ARRAY BOUNDS ERROR	Computed element address in an array is not within total array boundaries
ARRAY DECLARE ERROR	Computed array size is negative
ARRAY DIMENSION ERROR	Array used as an actual parameter in a procedure call has a different number of dimensions from the array specified as a formal in the procedure declaration
BOOLEAN INPUT ERROR	In Boolean formats F or P, the input character is not F or T, or 0 or 1
CHN xxxxxxxxxxxxxx	The number given defines channel on which the preceding error occurred
DISPLAY EXCEEDED	Block structure is nested to more than 32 levels; this error can only occur because of calls to pre-compiled procedures
EXPONENTIAL ERROR	Argument of the EXP procedure is too large
FLOAT TO FIX ERROR	Result of converting a normal floating-point number to fixed-point form exceeds 48 bits

Message	
FORMAT ITEM ERROR	More characters in expanded format item than permitted in INPUT, OUTPUT, IN LIST, and OUT LIST
FORMAT MISMATCH	A syntactically correct format string appears to be incorrect (probably machine or system malfunction)
FORMAT REPLICATOR	Replicator in a call to the FORMAT procedure not in the proper range
FORMAT STRING ERROR	Incorrect format string
GET/PUT ARRAY ERROR	GET ARRAY and PUT ARRAY may not be used on channel for which formatting and format area have been specified
H/V LIM ERROR	H LIM and V LIM arguments L, R and L', R' out of range
ILLEGAL CHANNEL CARD	Syntax of channel card is incorrect; the incorrect card is printed before the program is terminated
ILLEGAL IN-OUT	Illegal operation requested for equipment selected
ILLEGAL MODE CALL	T parameter in call to MODE procedure not in proper range
ILLEGAL STRING INPT	During a call of INPUT or IN LIST, an attempt is made to read into a string parameter
I/O CHANNEL ERROR	Normal input-output procedures (all except GET ARRAY and PUT ARRAY) cannot be performed on non-formatted channels
LAYOUT CALL ERROR	Procedures established by H END and V END and label set by NODATA are no longer accessible after return from the layout procedure called by IN LIST or OUT LIST
LOGARITHM ERROR	Argument to LN procedure may not be negative or zero
LOST DATA	Information lost during transmission because of a hardware malfunction
MANUAL INTERRUPT	Manual interrupt has occurred for which no provision has been made with MANINT procedure
NUMBER SYNTAX	A number input via standard format does not conform to proper syntax
NUMERIC INPUT ERROR	Data input under format control does not conform to numeric input format
OUT CHARACTER ERROR	Parameter in a call of OUT CHARACTER is not in proper range
PARAM COUNT ERROR	Incorrect number of actual parameters in procedure call
PARAM KIND ERROR	Kind of an actual parameter in a procedure call does not correspond to the kind of the associated formal

Message	
PARAM TYPE ERROR	Type of an actual parameter in a procedure call does not correspond to the type of the associated formal
SIN - COS ERROR	Argument to SIN or COS procedure is too large
SQUARE ROOT ERROR	Argument to the SQRT procedure may not be negative
STACK OVERFLOW	Data requirements of program exceed available memory
STANDARD OUTPUT ERR	Standard output can be used only for numeric and string formats
STRING ELEMENT ERR	Rules of STRING ELEMENT violated
SWITCH BOUNDS ERROR	Switch designator is out of the switch range
SYSPARAM - CHANNEL	SYSPARAM procedure can be called only for formatted channels
SYSPARAM - WRONG F	SYSPARAM procedure is called with incorrect F parameter
SYSPARAM - WRONG Q	SYSPARAM procedure is called with incorrect Q parameter
TABULATION ERROR	Argument of TABULATION procedure is not in proper range
UNASSIGNED CHANNEL	No channel defined for a channel number used in program
UNCHECKED EOF	End-of-file mark detected for which no provision has been made with EOF procedure
UNCHECKED PARITY	Uncorrectable parity error detected for which no provision has been made with PARITY procedure
UNDEFINED FOR LABEL	Attempt to jump into middle of a for statement
UNTRANSLATED IN ERR	In untranslated formats, I, R, L, and M; input field contains non-octal characters

OBJECT TIME I/O MESSAGE ON CTO UNIT

RE-LOAD LUN xx - PRESS GO WHEN READY

xx indicates device:

The card hopper is empty, refill hopper; press GO to continue.

The card punch has failed to feed. When corrected, press GO to continue.

Reload printer with paper; press GO to continue.

The end of tape has been encountered on an output tape; the tape has been backspaced, two end-of-file marks written on it, and unloaded. Mount a new tape on the designated unit; press GO to continue.

OBJECT TIME I/O MESSAGES ON STANDARD OUTPUT UNIT

ALGOL-I/O-ERROR yy ON LU xx

Uncorrectable error occurred during execution in segmented mode. The type of error is indicated by yy as follows

yy =	SG	System malfunction concerning segment tape control
	P1	Attempt to use segment tape
	P2	Attempt to use library tape (63)
	P3	Attempt to do other than write on 61 or 62
	P4	Attempt to do other than read on 60
	P5	Attempt to read past EOF on 60

PROGRAM EFFICIENCY HINTS E

Since the object code produced by any ALGOL language structure is not necessarily in direct correspondence to the apparent simplicity of the structure, the user can inadvertently produce object program ineffficiency by choosing one structure rather than another for a particular operation. Very often, the choice is not obvious.

In general, the user should avoid the complex structures and features of the ALGOL system (such as call-by-name).

The following explanations should help the user utilize the language as efficiently as possible and to decide which features, on balance, provide the most advantages for any operation. The list is representative and not meant to be exhaustive.

Compile Time

Compile time can be saved, in general, by any simplification of the program. However, no significant time is saved relative to the basic compilation time for the simplest statements.

Similarly, compile time can be saved in searching the identifier table if the identifiers are reduced in number or made as short as possible. Here again, the saving is relatively insignificant.

If there is a shortage of space at compile time, identifiers should be shortened and their number reduced until compilation is possible. Alternatively, a program can be divided into a main program and separately compiled self-contained procedures.

Object Time

Avoid Mixed-Mode Arithmetic

The evaluation of any expression involving variables not all of which are of the same type (not all real or all integer) results in one or more conversions from normal floating-point form to fixed-point form or vice-versa.

Use Integer Subscript Expressions

All subscripts are treated as integer variables, so that the use of real subscript expressions involves a conversion from normal floating-point form to fixed-point form.

Avoid Arrays where Simple Variables will Suffice

If all references to the elements of an array have integer constant subscripts, the elements of the array can each be declared as simple variables.

Elimination of an array saves the array declaration code, the code required to calculate the address of each element, and the stack space occupied by the entries which describe the array.

Minimize the Number of Dimensions in an Array

If all references to an array have integer constant subscripts for one of the dimensions, that dimension can be eliminated from the array by appropriate redefinition of the elements of the array into one or more simpler arrays.

Elimination of an array dimension saves part of the code required to calculate an element address in the array, and the stack space occupied by the entry which describes the dimension.

Declare All Arrays with the Same Bounds in the Same Declaration

If arrays with identical dimensions are declared together, each array still maintains its own stack entry and an entry for each of its elements; however, only one set of entries which describe the common dimensions is included in the stack.

Since fewer entries are included in the stack, not only is stack space saved but the declaration code for these arrays is reduced.

Declare Each Array, Switch, Label, and Procedure in Outermost Feasible Block

All such declarations produce object code which is executed whenever the block containing the declaration is entered. Since blocks at a higher level are generally entered fewer times than blocks at a lower level, the declaration code is executed fewer times if the declaration is made in the outermost feasible block rather than the block in which the array, switch, label, or procedure is used.

Avoid Call-by-Name

Each mention of a parameter called by name produces a procedure call in the object code. This becomes especially costly when the parameter called by name appears in code which is executed repetitively (such as within a for statement loop).

SAMPLE PROGRAM F

The following program is in the exact form that is punched into the cards that comprise the source deck (the hardware language)

```
2-DIMENSIONAL ARRAY.
THIS PROGRAM DECLARES A SERIES OF ARRAYS OF EVER-INCREASING
DIMENSION.  THE ARRAY IS THEN FILLED WITH COMPUTED VALUES, ONE
OF WHICH IS ALTERED.  THE ALTERED VALUE IS THEN SEARCHED FOR
AND PRINTED.
THE PROGRAM HALTS WHEN THE DECLARED ARRAY SIZE EXCEEDS THE
AVAILABLE MEMORY.  WHEN THIS OCCURS, THE PROGRAM EXITS WITH
THE MESSAGE            STACK OVERFLOW                ON THE STANDARD
OUTPUT UNIT.
'BEGIN' 'INTEGER' I.,
  I..=10.,
L..I..=I+1.,
  OUTPUT(61, '('/',3D')',I).,
  'BEGIN' 'ARRAY' A(/-3*I..-I,I..2*I/)., 'INTEGER' P,Q.,
    'FOR' P..=-3*I 'STEP' 1 'UNTIL' -I 'DO'
      'FOR' Q..=I 'STEP' 1 'UNTIL' 2*I 'DO'
        A(/P,Q/)..=-P+100*Q.,
    A(/-2*I,I+2/)..=A(/-2*I,I+2/)+10000.,
    'FOR' P..=-3*I 'STEP' 1 'UNTIL" -I 'DO'
      'FOR' Q..=I 'STEP' 1 'UNTIL' 2*I 'DO'
        'IF' A(/P,Q/) 'NOT EQUAL' 100*Q-P 'THEN'
        'BEGIN' OUTPUT(61,'('/',5D')',A(/P,Q/)) 'END'.,
    'GOTO' L
  'END'
'END'.,
          'EOP'
```

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET

3100/3200/3300/3500 COMPUTER SYSTEMS

ALGOL Reference Manual

Pub. No. 60134800 February, 1966

YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM NAME: _____

BUSINESS ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

FOLD

FOLD

31/32/33/3500 ALGOL REFERENCE MANUAL

CONTROL DATA SALES OFFICES

ALAMOGORDO • ALBUQUERQUE • ATLANTA • BILLINGS • BOSTON • CAPE CANAVERAL • CHICAGO • CINCINNATI • CLEVELAND • COLORADO SPRINGS DALLAS • DAYTON • DENVER • DETROIT • DOWNEY, CALIFORNIA • GREENSBORO, NORTH CAROLINA • HONOLULU • HOUSTON • HUNTSVILLE • MIAMI MONTEREY, CALIFORNIA • INDIANAPOLIS • ITHACA • KANSAS CITY, KANSAS LOS ANGELES • MADISON, WISCONSIN • MINNEAPOLIS • NEWARK • NEW ORLEANS • NEW YORK CITY • OAKLAND • OMAHA • PALO ALTO • PHILADELPHIA • PHOENIX • PITTSBURGH • SACRAMENTO • SALT LAKE CITY SAN BERNARDINO • SAN DIEGO • SANTA BARBARA • SAN FRANCISCO SEATTLE • ST. LOUIS • TULSA • WASHINGTON, D. C.

AMSTERDAM • ATHENS • BOMBAY • CANBERRA • DUSSELDORF • FRANKFURT • HAMBURG • JOHANNESBURG • LONDON • LUCERNE • MELBOURNE MEXICO CITY • MILAN • MONTREAL • MUNICH • OSLO • OTTAWA • PARIS TEL AVIV • STOCKHOLM • STUTTGART • SYDNEY • TOKYO (C. ITOH ELECTRONIC COMPUTING SERVICE CO., LTD.) • TORONTO • ZURICH

CONTROL DATA CORPORATION

8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

Pub. No. 60134800

Litho in U.S.A.