# APLGOL a Structured Programming Language for APL

Harwood G. Kolsky, IBM Scientific Center, Palo Alto, California, USA

## ABSTRACT

APLGOL is a language providing interstatement control structure for APL. It permits programs to be written using the power and conciseness of standard APL expressions in conjunction with structured programming concepts to emphasize more of the overall program control flow, rather than the details of individual statements.

The APLGOL System described consists of three parts: an Editor, an APLGOL-to-APL compiler and an APL-to-APLGOL reverse compiler. All three parts are themselves written in APL.

## 1. INTRODUCTION

The idea for APLGOL arose during the Fall of 1971 when John R. Walters of the IBM Palo Alto Scientific Center was teaching a class in computer science at Stanford University using APL instead of the usual ALGOL-W. The class observed that although an algorithm written in APL may be considerably shorter and more concise than the same algorithm written in PL/I or ALGOL, APL required explicit interstatement control to be written.

Although APL contains a great number of elegant operators, these operators are designed to manipulate scalar, vector, or array data. Only a single branch (right arrow) is available to handle whatever interstatement control is necessary. This is roughly equivalent to a machine-oriented conditional branch instruction, which is, of course, very general but as a consequence, the control flow within an APL program can be obscure and non-structured.

It is the property of APL which has led to statements being made that APL is "hostile" to structured programming. We feel that the directness with which the APLGOL system has been implemented to co-exist with APL certainly disproves this claim. APL when augmented by APLGOL is now one of the best structured programming languages.

Robert Kelley, one of Walter's students and co-workers, wrote the first APLGOL compiler in APL, publishing his results in 1972 and early 1973 (Refs. 5,6). One of the major considerations in designing APLGOL has been to replace the branch arrow with more descriptive keyword-oriented structures to clarify interstatement control. The first version of APLGOL was an attempt to provide either ALGOL-like or PL/I-like control structure syntax for a common set of semantics.

During 1973 Kelley and Walters (Ref. 7) revised and augmented APLGOL by adding the concept of a reverse compiler to produce APLGOL from compiled APL. In the original version a character form of the source text had to be maintained along with the object APL procedures. In the 1973 version the reverse compiler was used to recreate the character source text in a canonical structured form as required, so that only one form of the program was necessary for maintenance, listing, and execution.

The evolution of APLGOL syntax and semantics has resulted in the addition of a rich assortment of control structures including IF, WHILE, UNTIL, FOREVER, FOR, and CASE statements.

Finally, the whole topic of labels and branches has been examined, especially with respect to the control structures and structured programming techniques. The result has been to eliminate the GOTO statement and its attendant labels in favor of LEAVE, ITERATE, and RESTART statements, which may only access specific points within the scope of the control structure nest. APLGOL has thus become a truly GOTO-free, and label-free language. A version of this APLGOL written in APL was published earlier this year. (Ref. 8). Specific details given in this paper refer to that system.

## 2. The APLGOL Language

APLGOL is a language for providing a structured program interstatement control for APL. The APLGOL Language is oriented to an APL interpreter as the target machine. In general, APL semantics are unaltered in APLGOL, and only minor changes occur in the syntax. In APLGOL the comment delimiter, ⍝ , for example, must appear at both ends of a comment. Additionally, the semicolon used in APL for catenation has been replaced by the union symbol, ∪, in APLGOL.

An APLGOL program contains statements and comments arranged to describe the program's execution. The set of tokens describing an APLGOL program may be either basic symbols or APL expressions. The lexicon for APLGOL is identical to the APL character set. For details concerning the APL character set and APL expressions, see the APL Language description (Ref. 1). The basic symbols for APLGOL are single characters and reserved words as follows:

; : ⊂⊃∪⍝ A B C D E F I L N O P R S U W X DO IF OF END FOR CASE ELSE EXIT NULL STEP THEN BEGIN LEAVE UNTIL WHILE ASSERT REPEAT FOREVER ITERATE RESTART SUBCASE PROCEDURE

Comments are a sequence of zero or more characters enclosed with the comment delimiter character, ⍝ . Comments may appear anywhere in an APLGOL program but maynot be imbedded in a basic word or APL character string; it is understood that they do not affect the execution of a program.

Elementary constructions are syntactic rules of the following form:

< LEFT SIDE > ::= < RIGHT SIDE TOKEN SEQUENCE >

where the left side is a single token which may replace the sequence of one or more tokens on the right. English words in the brackets and are often used to describe approximately the nature of the tokens. By collecting and substituting a right hand token sequence the proper left side token for it and then collecting further right side tokens with further substitution, etc., a sequence of syntactic rules can be obtained for an entire APLGOL program. The collection of each elementary action or semantic rule applied for each substitution of a left hand side for a right hand side produces the resultant APL object program.

## 2.1 APLGOL Statements

APLGOL contains several types of statements, which are either basic statements executed independently, or control statements, which govern the execution of other basic or control statements.

Basic Statements

APL Statement
EXIT Statement
Empty Statement
NULL Statement
ASSERT Statement
ASSERT Level Statement

Control Statements

IF Statement
BEGIN Block
WHILE Statement Prefix
FOR Statement Prefix
FOREVER Statement Prefix
REPEAT Block
CASE Block
LEAVE Statement
ITERATE Statement
RESTART Statement

## 2.1.1 Basic Statements

The most fundamental APLGOL statement is the APL statement, written as an APL expression terminated by a semicolon. This statement may contain any valid combination of APL operations and operands used to form a line of APL code, excluding the branch instruction (right arrow).

The EXIT statement causes an exit from the current procedural level to the next outer level. It may optionally contain an expression to be evaluated just prior to the exit.

The Empty statement in APLGOL permits the programmer to write partial programs which are correct both syntactically and semantically. Should the statement be executed, the comment associated with the Empty statement is printed, and the execution is halted. From that point the programmer may choose whatever course of action he wishes. Mainly, the Empty statement is useful when a programmer wishes to debug certain portions of his code while other parts have not yet been written.

The NULL statement expresses a null action, and is primarily useful in conjunction with CASE blocks.

The ASSERT statement is useful for developing programs.

The integer specified in the first part of the statement is checked at compile time with a parameter set by the programmer. If the value of the integer is less than the parameter, the assertion test is not compiled into the program. The relational expression in the second part of the statement allows a programmer to make assertions about the correctness of his program. The assertion expression is evaluated dynamically during the program's execution. If the test fails, the program is halted following a message printing 'assertion fails.'

The assertion level parameter set by the programmer is valid for the current block and inner block levels, unless another assertion level is specified at an inner nesting level. When the current block is completed, the assertion level pertaining to the next outer block level is restored. If the programmer does not initially specify an assertion level, a value of 10 is assumed.

## 2.1.2 Control Statements

Several statements are used in APLGOL to control the execution of other statements. In general, they specify conditional, iterative, or selective statement execution.

The IF statement conditionally executes a subsequent statement.

Optionally, it may contain an ELSE clause to choose between two alternate statements for execution.

Since many of the statements in APLGOL, such as the IF, control the execution of a single statement, a BEGIN block is available to group several statements into a single unit. The BEGIN block uses a matched pair of BEGIN and END keywords. The semicolon on the true part statement is omitted if an ELSE clause is present.

The WHILE, FOR, and FOREVER statement prefixes may be used to specify how to execute an attendant statement. A BEGIN block is often useful in conjunction with these prefixes, enabling them to control the execution of an entire group of statements.

The WHILE statement in APLGOL permits iterative execution of a subsequent statement. The relational expression specified in the WHILE statement header is tested each time before executing the statement. In order for the test to fail, the values in the relational expression must be changed as a result of the statement's execution.

The FOR statement appears in two possible forms: The FOR...UNTIL...DO form and the FOR...UNTIL...STEP...DO form. The first expression must contain an assignment to initialize the induction variable for the statement. The step for the iteration may be specified as shown in the more complex form. Otherwise, a step of one is implied. The test for the iteration is performed at the top of the block each time before executing the statement.

A FOREVER statement is used in APLGOL for unconditional and continuous iteration of a statement. An escape is caused typically by an EXIT or a LEAVE, although an ITERATE or a

RESTART of an outer statement will also terminate a FOREVER statement.

A REPEAT statement is used for repetitive execution of a statement block. It is similar to the WHILE, except the condition is tested at the end of the block. Consequently, the statements contained within the REPEAT block are executed at least once. A REPEAT statement also differs from the WHILE in its format.

A CASE block is used to select a particular statement in the block for execution. The expression in the CASE header produces an index value specifying which subcase statement is to be executed. The integer in the second part specifies the maximum subcase index value, with 0 as the lowest value. This is true independent of the APL origin being used.

A subcase statement may be any type of basic statement or control statement. Each subcase statement in the subcase list is preceded by an integer followed by a colon to identify the particular subcase. As a consequence, subcases may be written in an arbitrary order, and null cases may be designated by omitting the subcase expression, although a NULL statement may be used explicitly to specify null subcases. Additionally, several subcase expressions may refer to the same subcase.

## 2.1.3 Leave, Iterate and Restart Points

Statements in a structured procedure lie within some nest of control statements, and for each control statement a LEAVE, ITERATE, and RESTART point has been defined. These points can be accessed from within the nested structure by LEAVE, ITERATE, and RESTART statements which use a control statement list to reference the particular control statement. Consequently, branches are restricted to preserve the disciplines of structured programming.

The LEAVE statement causes control to resume with the first statement following the specified control statement.

The control list contains combinations of the following reserved words to designate the most immediate control structure in the nest which satisfies the pattern of the list:

IF FOR CASE WHILE REPEAT FOREVER SUBCASE PROCEDURE

The same control list serves the LEAVE, ITERATE, and RESTART statements. If the control list designates a control structure which does not contain the particular LEAVE, ITERATE, OR RESTART, the statement is adjusted for the procedure level. The figure containing the example program shows some examples of the control list in conjunction with the LEAVE statement.

In the first example of the LEAVE, on line 17, the control list designates the on line 2 by locating the nearest outer REPEAT in the control structure nest, and then the nearest outer WHILE statement from the REPEAT. Should this LEAVE statement be executed, control would resume with line 33, the first statement following the complete WHILE statement. The control list in the second LEAVE statement on line 21 refers to the WHILE on line 6. Effectively, control would resume on line 30 if this LEAVE statement were executed. The third LEAVE statement refers to the WHILE on line 11, and would cause control to resume on line 27 if it were executed.

An ITERATE of a REPEAT implies another iteration over the statements in the block if the condition specified in the UNTIL clause is valid. A RESTART resumes control at the entrance of the REPEAT block without testing the condition.

An ITERATE of a FOR statement begins another iteration if the induction variable has not passed the FOR statement limit after adding the proper step. A RESTART begins the entire FOR statement again, including the initialization of the induction variable.

RESTART and ITERATE both denote identical actions when used in conjunction with the IF, WHILE, CASE, SUBCASE, PROCEDURE, and FOREVER statements. They cause each such statement to be re-executed. (See the Figures for diagrams of the LEAVE, ITERATE, and RESTART points in each of the control structures.)

## 3. The APLGOL System

The standard APL System operates either in a computational mode or a procedure-definition mode. In this latter mode an APL procedure may be created or modified as desired. When this has been accomplished, and the computational mode is entered, the string of characters representing the function is encoded (or compiled,) into an internal form more suitable for the APL interpreter. To edit this procedure, the internal form is transformed back to the character form as the programmer changes again from computational mode into procedure-definition mode. As a result, a workspace contains only a single copy of a procedure.

In the APLGOL System a special APLGOL editor, similar to the CMS editor (Ref. 4), is available for the programmer to create and edit APLGOL procedures. The programmer may invoke one of a pair of compilers, either to translate APLGOL source programs into internal APL object programs, or to translate internal APL object programs back into APLGOL source programs for subsequent editing. (See Fig. 1)

To invoke the APLGOL editor the programmer types:

```
< NAME >← EDIT  < NAME >
```

where the name specifies a character array containing the representation of an APLGOL procedure. When editing is complete, the EDIT program returns the updated APLGOL program as a character array.

The EDIT commands are listed below in section 3.1.

To evoke the compiler one types

```
APLGOL < NAME >
```

The APLGOL compiler produces an APL function whose name is < APLFN > given by the PROCEDURE < APLFN >; statement in the APLGOL source.

The APL interpreter only operates on the indistinguishable internal form, which may have been produced from standard APL or APLGOL. (It should be noted, however, that although it is possible to print and edit APLGOL programs using the standard APL editor, it is not possible to produce APLGOL programs from arbitrary APL programs. The Reverse APL to APLGOL compiler relies heavily on the canonical form of the APL program as it is compiled from APLGOL.)

To use the reverse compiler one types

```
REVERSE'< APLFN>'
```

The result is an APLGOL source program in expanded, indented form. It is left in a global character array variable named "OUT".

The APLGOL System was designed recognizing that the user requires different human factors capabilities when he is typing or editing a program than when he lists a program to display its structure. Consequently, the APLGOL compiler has been designed to accept text with combinations of abbreviated and completely spelled keywords and many source statements to the line, while the REVERSE compiler produces source programs with fully spelled keywords in statements, one statement per line, with two-space indenting for each layer of nesting. Thus, a procedure can be entered as:

```
P SAMPL; I A≠B T B A←C; F J←I U ⌊(N-1 +I)÷2 D
B L2←L3←L4L-J-1; L4← -L-1↓ρY←7 DYADF L; E; E E A←D;
I 2=ρρZ T X C[2]←(1↓ρZ)÷N; Z←N,C,N,P,Q,R; E P
```

while it is produced by the reverse compiler and used for subsequent editing as:

```
PROCEDURE SAMPL;
  IF A≠B THEN
    BEGIN
      A←C;
      FOR J←I UNTIL ⌊(N-1+I)÷2 DO
        L2←L3←L4L-J-1;
        BEGIN
          L4←-L-1↓ρY←7 DYADF L;
        END;
    END
  ELSE
    A←D;
  IF 2=ρρZ THEN
    EXIT C[2]←(1↓ρZ)÷N;
  Z←N,C,N,P,Q,R;
END PROCEDURE
```

## 3.1 The APLGOL Editor

A single context-oriented editor is used to enter new programs, edit old ones, and list programs. This editor has been patterned after the CMS editor in VM/370 (Ref. 4). Compared with typical APL editors, this editor is keyed to context and not to line numbers. Its functions include locating the next occurrence of a character pattern, changing one character pattern to another, inserting, deleting, replacing, or printing lines of text relative to an implied cursor, and, finally, moving the cursor up or down a few lines or to the top or bottom of the text. The text has no special relation between APLGOL statements and physical lines; an arbitrary number of lines may be used to enter an APLGOL statement, or many statements can be entered on a line.

APLGOL source programs are actually APL character arrays. Keywords all begin with an underlined first letter for easy recognition, while other letters are not underlined in order to reduce keystrokes. Also, when programs are being entered, only the first underlined letter need be used. This is intended to facilitate the entering of programs and to reduce misspelling of keywords.

To create an entirely new source program the editor is invoked by entering:

```
NEWPGM←EDIT (0,N)ρ' '
```

where NEWPGM is the name of the new source text, and N specifies the line width. The editor is then in Command Mode and may accept commands for inserting, deleting, or changing lines of source text. To edit a text array, the edit procedure is invoked according to the following example:

```
ARRAY←EDIT ARRAY
```

where ARRAY is the name of both the old text being edited and the newly edited text. Different names can be used for the old and new texts if desired.

The programmer may terminate editing either by typing "FILE" to translate the APLGOL source text into internal APL, or by typing "QUIT" to abort the editing process without affecting the prior status of the procedure.

The EDITOR accepts the following commands:

INSERT .... or I ....

Insert text .... after the current line.

When the editor is in Command Mode, Insertion Mode is entered by typing an "I" followed by a carriage return. Following this, successive lines of text may be inserted by typing them one after the other as they are to appear in the text. To leave Insertion Mode and return to Command Mode, a null line is entered by pressing the carriage return on a new line before any other character.

DELETE n or D n

Delete n lines beginning with the current line. If n is omitted, then 1 is assumed.

PRINT n or P n

Print n lines beginning with the current line. If n is omitted, 1 is assumed.

NEXT n or N n

Step forward n lines in the Text. If n is omitted then assume 1.

UP n or U n

Step up n lines. If n is omitted then assume 1.

TOP or T

Position line pointer at first line in text.

BOTTOM or B

Position line pointer at last line in text.

LOCATE /..../ or L/..../

Search the lines following the current line for the line containing the text string .... The characters / and / are used to delimit the argument of locate; they are not part of the string being searched for. Any character may be used as a delimiter. If the pointer is on the last line of the function, the search will begin at the top of the function.

CHANGE /text1/text2/ or C /text1/text2/

Search the current line for text1, and replace it by text2 if it occurs. As noted, above, the character / serves as a delimiter and may be replaced by any character. If text1 is null, text2 is inserted at the beginning of the line. If text2 is null, text1 is deleted.

REPLACE .... or R ....

Replace the current line by the text .... If no text is given, this is the same as DELETE, UP, INSERT combined.

FILE or F

Leave the editor and assign the newly edited text to the target array specified.

QUIT or Q

Leave the editor and make no change to the status of the function. If the function was undefined prior to editing, it will still remain undefined. If it was a defined function, its definition will be unaltered.

Note: If the argument to NEXT, PRINT, LOCATE, or DELETE is such that the line pointer would move past the end of the function, then the line pointer is set to point to the last line. Similarly, if the UP command tries to point beyond the top line, the line pointer is set to point to the first line in the text.

## 3.2 The APLGOL Compiler

The compiler is organized into three main sections: (1) a syntax scanner, (2) a lexical scanner, and (3) an APL object text generator. One of two driving procedures is used to initialize the tables for the appropriate syntax before invoking the syntax scanner.

The syntax scanner is the controlling procedure, obtaining meta symbols from the lexical scanner, identifying the grammar rules, and then invoking the text generator as necessary to produce the appropriate APL object text.

On each invocation, the lexical scanner returns a single meta symbol number representing a reserved word, label, special character, or an APL expression. Source text characters are scanned for one of a very limited set of characters, which is then used to key a detailed search for a specific meta symbol; e.g., the first character in each reserved word is underlined to aid in distinguishing it as a particular meta symbol, both for the lexical scanner and for the reader. The terminal meta symbols listed in Appendix II are detected in the lexical scanner.

When a grammar rule is identified by the syntax scanner, the text generator is invoked with a number corresponding to that grammar rule to locate the applicable portion of the generator. Information accumulated in the compile stack and elsewhere is then used to generate labels, branches, or to produce a single APL text line from a buffered APL expression.

An output procedure is employed to form an array of object text, and is invoked from the text generator each time a new line is created. This object text array remains in the compiler workspace, so that extraneous branches and labels can be removed after all the object text has been produced.

The semantics of the control structures are indicated in the enclosed figures.

In the APL object text produced by the simple one-pass compiler, many of the labels and branches may be unnecessary, particularly those statements which consist of a label and an absolute branch. To remove them, the compiler builds tables as the object code is formed showing labels and their references. After all the object code has been produced, a label followed by an absolute branch can be detected and removed with references to that label suitably revised to references to the target of the absolute branch. Further, if multiple labels appear on a single line of object text, all but one are removed, and the references are changed accordingly.

## 3.3 The APL to APLGOL Back-Compiler

It has been customary in APL systems to retain a single internal form for each APL procedure and to translate this to or from a character representation, as necessary. This requires a pair of translators, one for each direction.

Although APLGOL and APL are rather dissimilar, it has been possible to construct both translators. The translation from APL to APLGOL, however, is heavily conditioned by the patterns produced by translation the other way. The main advantage of the single form is that there are not multiple forms of the source program to get out of synchronization. In the original APLGOL implementation (Ref. 5) the APLGOL source was a separate entity from the APL object, and one could not guarantee that both forms were at the same maintenance level. Additionally, storage requirements are reduced by having only a single copy of a procedure.

The reverse translation from APL to APLGOL will produce a stylized canonical form, which may appear quite different from the original APLGOL program as entered. Rather than attempt to duplicate the original, the translator was designed to produce an indented format which displays the structure of the program graphically. In this form, keywords are fully spelled, and each control level is consistently indented, typically with one statement per line. This graphical representation of the program is very useful to depict the program structure. Because this form is available for listing an APLGOL program that has been successfully translated to APL, the APLGOL program can always be listed in a standard format, thereby promoting a consistent style.

In general the reverse compiler does not change the structure of the APLGOL program. It does, however, keep count of the number of APL statements in each block. If the APL function is edited to have two or more statements where only one was present in the original APLGOL source, the reverse compiler will add a BEGIN ...END pair around the statements during back compilation.

## 4. ACKNOWLEDGEMENTS

The author wishes to acknowledge the work of the IBM Palo Alto Scientific Center staff for their inspired work in designing, implementing and testing the APLGOL system. The major original contributions were made by Robert A. Kelly, John R. Walters and Dan McNabb (See Refs. 5-7). Special thanks are due to H. Joseph Myers for the more recent modifications and testing APLGOL under many conditions.

The author is particularly grateful to John R. Walters for making available to us his most recent copyrighted APLGOL functions written in APL. (Ref.8)

## APPENDIX I: References

1. APL/360 User's Manual, GH20-0683, IBM Corporation.
2. Dijkstra, E. W., "The Humble Programmer", 1972 Turing Lecture, Communications of the ACM, Vol. 15, No. 10, October 1972.
3. Dijkstra, E. W., "Structured Programming", (with O. J. Dahl and C. A. R. Hoare) Academic Press, London, October 1972.
4. IBM Virtual Machine Facility/370: Edit Guide, GC20-1805, IBM Corporation.
5. Kelley, R. A., "APLGOL, a Structured Programming Language for APL", IBM Palo Alto Scientific Center Report No. 320-3299, August 1972.
6. Kelley, R. A., "APLGOL, an Experimental Structured Programming Language", IBM Journal of Research and Development, Vol. 17, No. 1, January 1973.
7. Kelley, R. A. and Walters, J. R., "APLGOL-2 A Structured Programming System for APL", IBM Palo Alto Scientific Center Report No. 320-3318, August 1973.
8. Kelley, R. A. and Walters, J. R., "APLGOL-2 A Structured Programming Language System for APL". Proceedings of APL-VI Conference, May 14-17, 1973.
9. Knuth, D. E., "A Review of Structured Programming" Stanford University Computer Science Department, STAN-CS-73-371, June 1973.

# *APPENDIX II: APLGOL SYNTAX*

```
[1]    <PROGRAM> ::= _|_ <PROCEDURE> <STATEMENT LIST> <END>
                     PROCEDURE _|_

[2]    <PROCEDURE> ::= PROCEDURE <EXPRESSION> ;

[3]    <STATEMENT LIST> ::= <STATEMENT> ;
[4]          | <STATEMENT LIST> <STATEMENT> ;

[5]    <STATEMENT> ::= <COMMENT LIST> <STATEMENT-A>
[6]          | <STATEMENT-A>

[7]    <COMMENT LIST> ::= <COMMENT STATEMENT>
[8]          | <COMMENT LIST> <COMMENT STATEMENT>

[9]    <STATEMENT-A> ::= <EXPRESSION>
[10]         | <ASSERT HEAD> : <EXPRESSION>
[11]         | <ASSERT HEAD>
[12]         | <EMPTY STATEMENT>
[13]         | NULL
[14]         | <BRANCH> : <CONTROL LIST>
[15]         | EXIT
[16]         | EXIT <EXPRESSION>
[17]         | <BEGIN> <STATEMENT LIST> <END>
[18]         | <REPEAT> <STATEMENT LIST> UNTIL <EXPRESSION>
[19]         | <IF CLAUSE> <TRUE PART> <STATEMENT>
[20]         | <FOR EXPR> DO <STATEMENT>
[21]         | <WHILE HEAD> DO <STATEMENT>
[22]         | <FOREVER> DO <STATEMENT>
[23]         | <CASE EXPR> BEGIN <SUBCASE LIST> <END CASE>

[24]   <ASSERT HEAD> ::= ASSERT <EXPRESSION>

[25]   <TRUE PART> ::= <STATEMENT> <ELSE>

[26]   <IF CLAUSE> ::= <IF> <EXPRESSION> THEN

[27]   <FOR EXPR> ::= <FOR LIMIT>
[28]         | <FOR STEP>

[29]   <FOR STEP> ::= <FOR LIMIT> <STEP> <EXPRESSION>

[30]   <FOR LIMIT> ::= <FOR HEAD> UNTIL <EXPRESSION>

[31]   <FOR HEAD> ::= <FOR> <EXPRESSION>

[32]   <WHILE HEAD> ::= WHILE <EXPRESSION>

[33]   <CASE EXPR> ::= <CASE HEAD> OF <EXPRESSION>

[34]   <CASE HEAD> ::= CASE <EXPRESSION>

[35]   <END CASE> ::= <END> CASE

[36]   <SUBCASE LIST> ::= <SUBCASE>
[37]         | <COMMENT STATEMENT>
[38]         | <SUBCASE LIST> <SUBCASE>
[39]         | <SUBCASE LIST> <COMMENT STATEMENT>
```

## *APPENDIX II: APLGOL SYNTAX*

```
[40]  <SUBCASE> ::= <SUBCASE HEAD> <STATEMENT> ;

[41]  <SUBCASE HEAD> ::= <SUBCASE HEAD-1> :

[42]  <SUBCASE HEAD-1> ::= <EXPRESSION>
[43]          | <SUBCASE HEAD> <EXPRESSION>

[44]  <BEGIN> ::= BEGIN
[45]          | B

[46]  <BRANCH> ::= LEAVE
[47]          | ITERATE
[48]          | I
[49]          | RESTART
[50]          | R

[51]  <CONTROL LIST> ::= <CONTROL>
[52]          | <CONTROL LIST> <CONTROL>

[53]  <CONTROL> ::= REPEAT
[54]          | <IF>
[55]          | WHILE
[56]          | CASE
[57]          | SUBCASE
[58]          | PROCEDURE
[59]          | FOR
[60]          | FOREVER
[61]          | F
[62]          | R
[63]          | S

[64]  <END> ::= END
[65]          | E

[66]  <REPEAT> ::= REPEAT
[67]          | R

[68]  <ELSE> ::= ELSE
[69]          | E

[70]  <IF> ::= IF
[71]          | I

[72]  <FOR> ::= FOR
[73]          | F

[74]  <FOREVER> ::= FOREVER
[75]          | F

[76]  <STEP> ::= STEP
[77]          | S
```

***

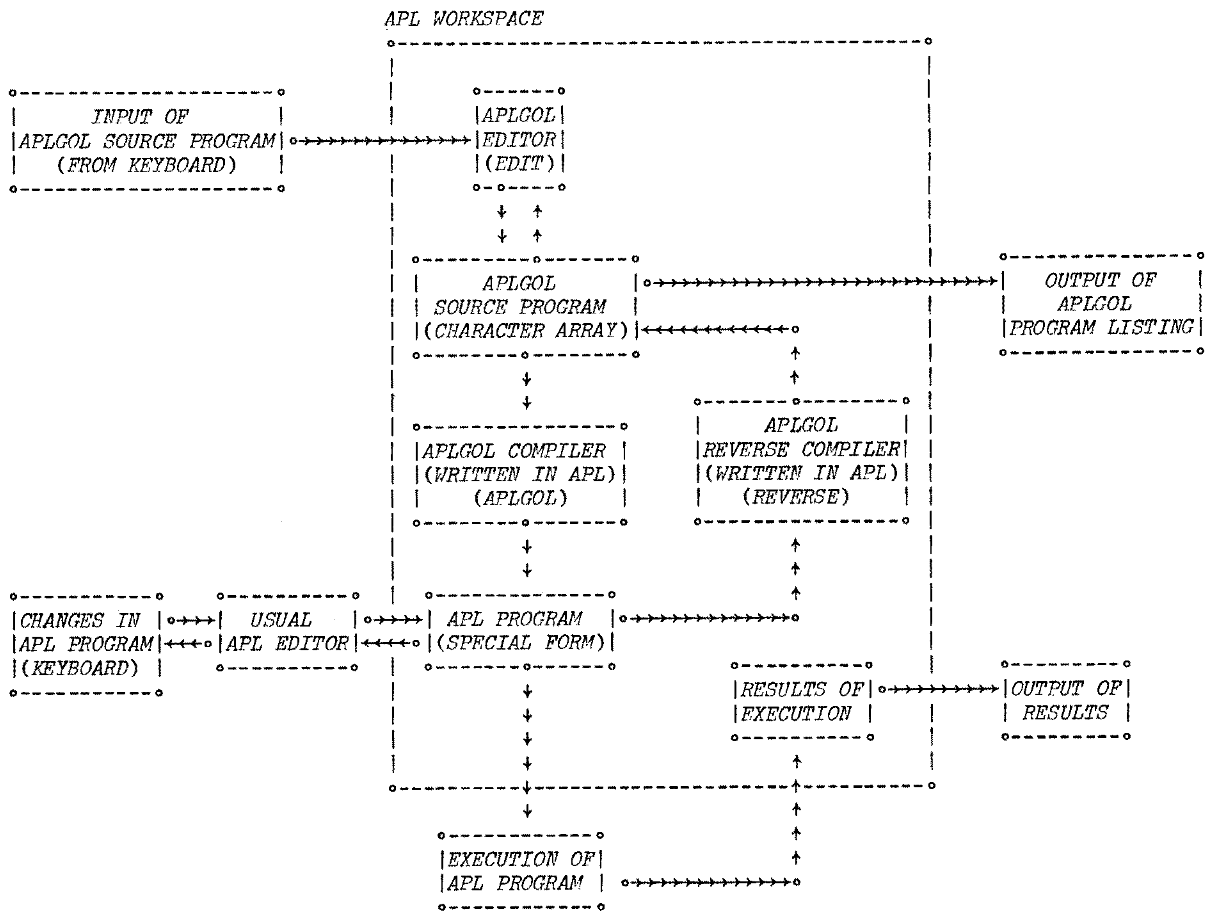

APL WORKSPACE
INPUT OF APLGOL SOURCE PROGRAM (FROM KEYBOARD)
APLGOL EDITOR (EDIT)
APLGOL SOURCE PROGRAM (CHARACTER ARRAY)
OUTPUT OF APLGOL PROGRAM LISTING
APLGOL COMPILER (WRITTEN IN APL) (APLGOL)
APLGOL REVERSE COMPILER (WRITTEN IN APL) (REVERSE)
CHANGES IN APL PROGRAM (KEYBOARD)
USUAL APL EDITOR
APL PROGRAM (SPECIAL FORM)
RESULTS OF EXECUTION
OUTPUT OF RESULTS
EXECUTION OF APL PROGRAM

Figure 1 *AN APLGOL SYSTEM WRITTEN IN APL*

## *APLGOL STATEMENTS*

### *BASIC STATEMENTS*

*APL STATEMENT*
*EXIT STATEMENT*
*EMPTY STATEMENT*
*NULL STATEMENT*
*ASSERT STATEMENT*
*ASSERT LEVEL STATEMENT*

### *CONTROL STATEMENTS*

*IF STATEMENT*
*BEGIN BLOCK*
*WHILE STATEMENT PREFIX*
*FOR STATEMENT PREFIX*
*FOREVER STATEMENT PREFIX*
*REPEAT STATEMENT PREFIX*
*REPEAT BLOCK*
*CASE BLOCK*
*LEAVE STATEMENT*
*ITERATE STATEMENT*
*RESTART STATEMENT*

*FIGURE 2*

```
<BASIC STATEMENT> ::= <EXPRESSION> ;
```

*EXAMPLES:*

```
     P←(2=+/[2]=S∘.|S)/S←ιN;
     'TIME= '∪T∪' RATE= '∪R∪' DISTANCE= '∪D[I←1+S;J←J|B];
```

```
<BASIC STATEMENT> ::= EXIT ;
                    | EXIT <EXPRESSION> ;
```

*EXAMPLES:*

```
     IF A=1 THEN
       EXIT
     ELSE
       EXIT P←P|5;
```

```
<BASIC STATEMENT> ::= ⊂ <EXPRESSION> ⊃ ;
```

*EXAMPLE:*

```
     IF SSECTODATE < SSMAX THEN
       ⊂ SOCIAL SECURITY COMP GOES HERE ⊃ ;
```

```
<BASIC STATEMENT> ::= NULL ;
```

*FIGURE 3*

```
<BASIC EXPRESSION> ::= ASSERT <EXPRESSION> : <EXPRESSION> ;
```

EXAMPLE:

```
     ASSERT 10 : A<ρTABLE;
```

```
<BASIC EXPRESSION> ::= ASSERT <EXPRESSION> ;
```

EXAMPLE:

```
     ASSERT 100;
```

FIGURE 4

```
<CONTROL STATEMENT> ::= IF EXPRESSION THEN <STATEMENT>
                       |IF EXPRESSION THEN <STATEMENT>
                            ELSE <STATEMENT>
```

```
<STATEMENT> BLOCK> ::= BEGIN <STATEMENT  LIST> END
```

EXAMPLES OF BEGIN BLOCKS IN CONJUNCTION WITH THE IF <STATEMENT> ARE THE FOLLOWING:

```
IF A>5 THEN          IF A>5 THEN          IF A≥5 THEN
  BEGIN                B←A|5;               BEGIN
    B←A|5;           ELSE                     B←1|5;
    A←C÷5;             BEGIN                  A←C÷5;
  END                    A←A+1;             END
ELSE                     C←B←B|C;         ELSE
  BEGIN                END;                 A←A+1;
    A←A+1;
    C←B←B|C;
  END;
```

FIGURE 5

```
<CONTROL STATEMENT> ::= WHILE <EXPRESSION> DO <STATEMENT>


<CONTROL STATEMENT> ::= FOR <EXPRESSION> UNTIL <EXPRESSION>
                              DO <STATEMENT>
                      | FOR <EXPRESSION> UNTIL <EXPRESSION>
                              STEP <EXPRESSION> DO <STATEMENT>


<CONTROL STATEMENT> ::= FOREVER DO <STATEMENT>
```

EXAMPLE:

```
     FOREVER DO
       IF FLAGFUNCTION THEN
         LEAVE : FOREVER;
```

```
<CONTROL STATEMENT> ::= REPEAT <STATEMENT LIST> UNTIL
                              <EXPRESSION> ;
```

FIGURE 6

```
<CONTROL STATEMENT> ::= CASE <EXPRESSION> OF <EXPRESSION>
                              BEGIN <SUBCASE LIST> END CASE ;


<SUBCASE> ::= <EXPRESSION> : <STATEMENT>
```

THE FOLLOWING IS AN EXAMPLE OF THE CASE STATEMENT BLOCK:

```
  CASE I|J OF 15
    BEGIN
      0:
        FOR K←I|J UNTIL (ρTABLE)[1] STEP J DO
          BEGIN
            TABLE[K;I]←FUN ARG;
            I←I+1;
            J←J-1;
          END;
      2:
        NULL;
      1:
        ⊂ CASE 1 EMPTY FOR NOW ⊃;
      10: 12: 14:
        BEGIN
          I←I-1;
          J←J-1;
          IF 10≥I|J THEN
            RESTART: CASE
          ELSE
            ITERATE: SUBCASE;
        END;
      5 :
        EXIT I,J;
    END CASE;
```

FIGURE 7

```
<BASIC STATEMENT> ::= LEAVE : <CONTROL LIST> ;
                    | ITERATE : <CONTROL LIST> ;
                    | RESTART : <CONTROL LIST> ;
```

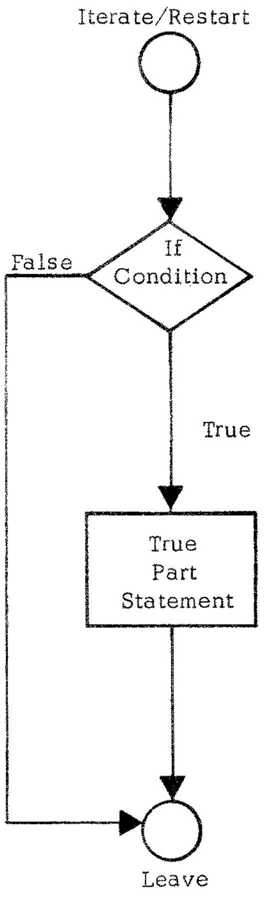
FIGURE 8

EXAMPLE:

```
PROCEDURE EX;
  WHILE A>B DO
    BEGIN
      REPEAT
        B←C[I];
        WHILE I<ρC DO
          BEGIN
            IF B=5 THEN
              BEGIN
                I←B;
                WHILE J<1↑ρD DO
                  BEGIN
                    IF 4<J←+/D[J;] THEN
                      BEGIN
                        A←A-1;
                        B←J;
                        LEAVE: REPEAT WHILE;
                      END
                    ELSE
                      IF B=10 THEN
                        LEAVE: WHILE WHILE
                      ELSE
                        IF J=ρC THEN
                          LEAVE: WHILE;
                    B←A←0;
                  END;
              END;
            I←I+1;
          END;
      UNTIL B>15;
      C←C,J,B;
    END;
  A←B;
END PROCEDURE
```

FIGURE 9

1. IF/THEN



2. IF/THEN/ELSE

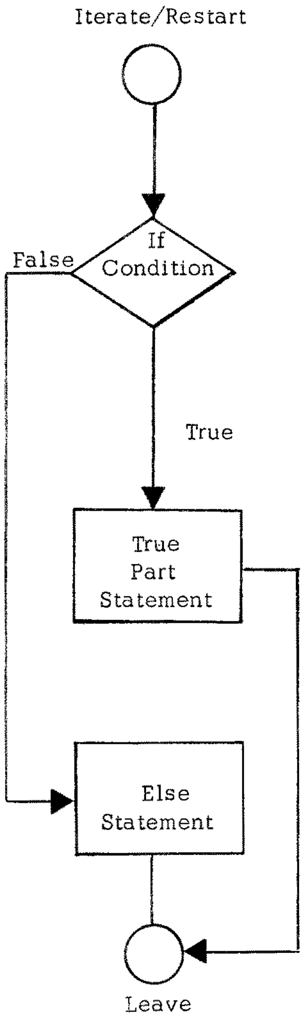
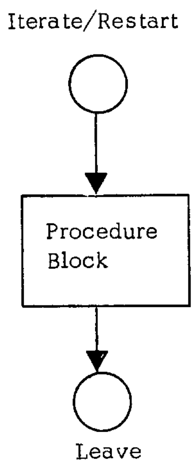


Figure 10 APLGOL

3. PROCEDURE BLOCK



4. FOREVER STATEMENT

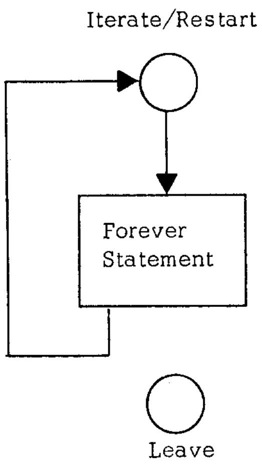
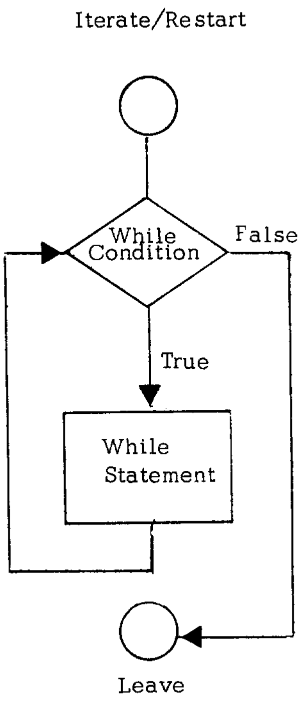


Figure 11

5. WHILE STATEMENT



6. REPEAT BLOCK

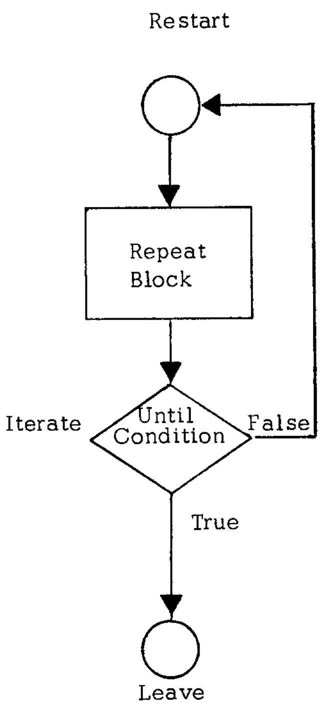
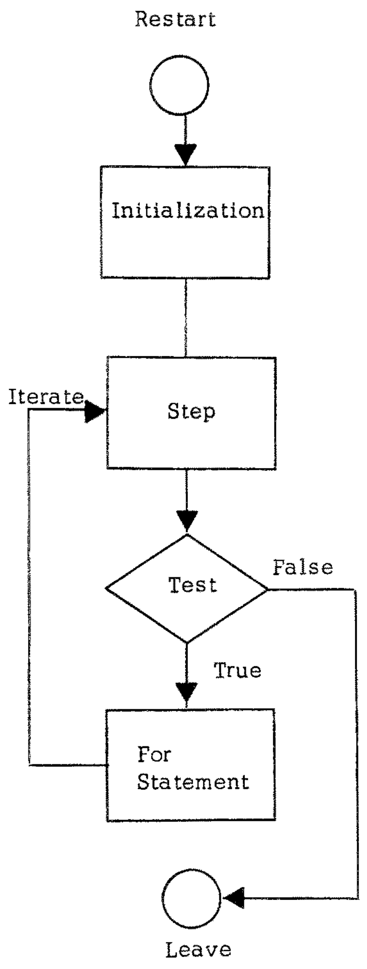


Figure 12

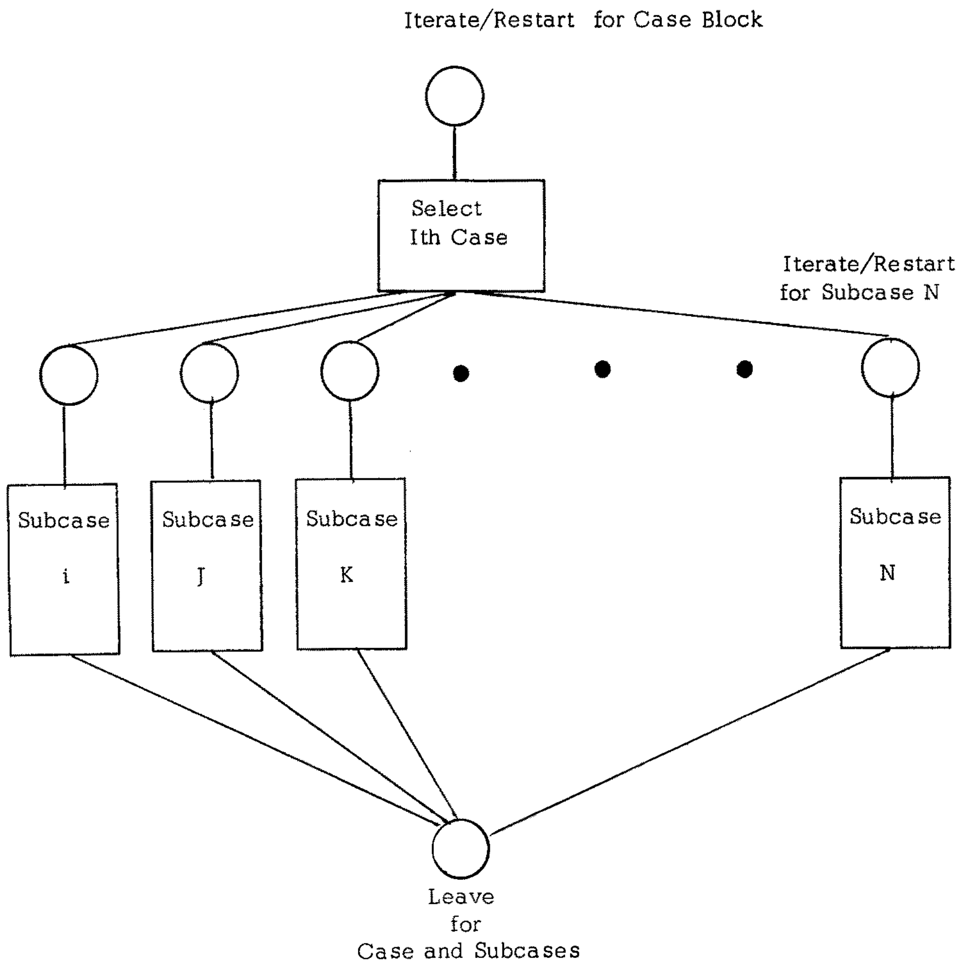
## 7. FOR STATEMENT



APLGOL

Figure 13

## 8. CASE BLOCK AND SUBCASE


Iterate/Restart for Case Block
Select Ith Case
Iterate/Restart for Subcase N
Subcase i
Subcase J
Subcase K
Subcase N
Leave for Case and Subcases

APLGOL

Figure 14