A METHOD OF COMBINING ALGOL AND COBOL

Jean E. Sammet
Data Systems Operations
Sylvania Electric Products
Needham 94, Massachusetts

SUMMARY

This paper presents a method for combining ALGOL and COBOL. The purpose and general approach is described; the basic principle is to try to give both groups what they want rather than forcing one group to conform to the other. The major and conceptual differences are listed and described under the headings (a) type of problem to be solved; (b) general usage of the language; (c) symbolism; (d) data description; (e) input-output. Four types and levels of interchangeability are listed and described, namely (a) transliteration of basic symbols; (b) transliteration of groups of symbols; (c) translation of groups of symbols; (d) arbitrary differences.

A listing of elements which are interchangeable under one of the first three categories cited above is given. Arbitrary differences between the two languages are given.

Two sections consider the changes which must be made to each language to have them coincide in the areas where they basically overlap. The paper ends with some indications of the magnitude of the modifications involved.

This paper is intended primarily to show a method which can be used to bring the languages together. The changes can only be made officially through the committee maintaining each language.

I. GENERAL DESCRIPTION OF METHOD

1. Purpose and General Approach

At this point in time, there exist two languages - namely, ALGOL and COBOL - which are designed to be problem oriented languages in the two major fields of computer applications: mathematics, and business data processing. (Some people prefer the terms "procedure-oriented" or "procedural" rather than "problem oriented". Since no definition of any of these terms has ever been agreed upon, the exact wording is not significant for the purposes of this paper.)

Although the need for combining the two languages seems fairly evident, it is worth pointing out a few of the advantages to be gained from such a step. The first - and most important - factor is that of cost. As long as computer manufacturers need to prepare two radically different compilers to satisfy their customers, the hidden "software" costs will continue to be quite high. Even such mundane matters as double training manuals and operating procedures add to the overall price the user pays for his equipment. Since there is a high degree of overlap (as will be shown later) it is certainly reasonable to consider bringing the languages together. A second factor is the desirability of extending the area of standard methods of writing programs. There are certainly large classes of problems for which COBOL is suitable and ALGOL is not, and conversely. A combination of the languages, tentatively named ALABOL (for Algorithmic And Business Oriented Language) would widen the class of problems which could be handled in the "standard" way. In no way is this meant to imply that ALABOL would serve as a universal panacea, or even as a single universal problem (or procedure) oriented language. However, experience from using such a language would be useful in narrowing the classical (and still existent) distinction between data processing and mathematical problems.

Both of those languages have many characteristics in common, not the least of which is that they both were developed by committees and have gained a fairly wide acceptance by the computing industry. However, their surface differences are certainly greater than their similarities. The main reason for this is the fact that there is virtually no practical intersection of the two committees. At the time this paper is written, the author is the only person who is officially a member of the two committees charged with the maintenance of the two languages. (This is not meant to imply that there are no other people interested in both languages simultaneously. Many of the people with the
dual interest have been unable to participate in both groups.) There appear to be two primary reasons for this lack of connection between the two groups. The first reason involves the basic concept of each language, and the area of problems for which it is intended. In general, those individuals interested in business data processing problems are not interested in mathematical and engineering problems, and conversely. However, the increasing number of problems which cut across both lines makes the amalgamation of these areas more prevalent and necessary. It is becoming much more difficult to retain the old individual compartments for large problems, and this fact must be reflected in language development.

The second reason for the lack of connection between the two groups is a certain amount of displeasure on the part of each with the approach taken by the other. These conceptual differences are indicated in a later section. However, this paper shows explicitly the large amount of common ground which actually exists. Furthermore, the purpose of the work reflected in this paper is to try to give both groups what they want, rather than forcing one group to conform to the other. The result might be considered by some to be a hodge-podge, but the author prefers to think of it as an effective compromise in which the computing industry gains a great deal, and comparatively little unhappiness is generated.

The present relationship between ALGOL and COBOL can be summarized by the following diagram:

The three types of shading represent three types of interchangeability and are discussed in a later section. The actual verbal meaning of the diagram is that there are many points in common between ALGOL and COBOL (where "in common" means that there is a 1-1 correspondence between certain elements). Thus, within the area of intersection, there are at worst some small notational differences which can trivially be resolved. Furthermore, even in the non-intersecting areas, there is a wider range of agreement than a lack of intersection would normally indicate.

The basic goal of this paper is to show a way in which the essential characteristics of both languages can be maintained. This work is not intended to develop a "universal language", although that might well be a byproduct. The totality of work involved in combining these languages is necessarily split into a few stages. Only the first two are being described here, but the others will be mentioned briefly to show the direction of future work. The first phase involves a clearcut statement of the major conceptual differences. Obviously this is essential if detailed technical work is to be done. The second phase, which is also covered in this paper, involves only those areas of both languages where they attempt to do the same type of operation. Later work will expand the area of concern. The general approach can be summarized by saying that wherever possible use will be made of the types of interchangeability described below in §1.2 and significant restrictions, changes, and additions will only be made where absolutely necessary.

In order to allow effective judgment and evaluation of this paper, it must be pointed out that there are several things it does not claim to do. First - and most important - it does not present a complete set of specifications for a new language. In some areas, it has been possible to show all the details, but in others, the magnitude of the work involved is too large to make this worthwhile unless strong support is shown.

Secondly, problems of implementation are not considered here, although nothing has been done to make the lot of the implementor any more difficult.

2. Major and Conceptual Differences

There are five major differences between ALGOL and COBOL. These can briefly be stated as:
From the collection of the Computer History Museum (www.computerhistory.org)

(a) Type of Problem to be Solved

It is probably a truism to state that ALGOL is primarily concerned with mathematical type problems, and in particular, with expressing algorithms. To quote from the ALGOL 60 report, "The purpose of the algorithmic language is to describe computational processes". On the other hand, the COBOL 60 report states that "The task of the committee was that of preparing a common business language. By this is meant the establishment of a standard method of expressing solutions for a certain class of problems normally referred to as business data processing".

(b) General Usage of the Languages

One of the major points of difference between the two languages is the implied (and sometimes stated) way in which the languages will be used. This is virtually equivalent to the amount of connection with computers each language possesses. More specifically, ALGOL was defined, and is being used primarily, as a communication language. That is to say, the emphasis has been on the establishment of a language which could be used to transmit algorithms and solutions of mathematical type problems. This concept is best illustrated by the use in ALGOL of a reference and a publication language as well as a hardware representation. The emphasis on the first two makes it clear that the actual running of these problems on a computer is considered by many - but not all - ALGOL experts to be a secondary matter. COBOL, on the other hand, has placed a major emphasis on its use in solving specific problems on a variety of computers. That is, COBOL experts are primarily concerned in making sure that the problems can be physically run on more than one computer and get the same answers. The lack of anything but a set of hardware characters (and the limitation of the number of these to 51) is clear proof of the strong computer connection.

(c) Symbolism

There are several major differences in symbolism between the two languages. The first of course is the obvious one of having mathematical notation wherever possible in ALGOL, as opposed to the use of normal English wherever possible in COBOL. A second difference involves the establishment of three levels of languages in ALGOL, of which the reference language is the only standard one. The full effect of this is merely a reflection of the point cited in (b) above. Thus, if people are primarily interested in exchanging algorithms, then mundane problems such as non-existent symbols can be ignored. If, on the other hand, the primary purpose is to run problems on a computer, then character sets and hardware limitations must be taken into careful consideration. A third difference is a philosophic one of whether or not the reader should be forced to work very hard. In ALGOL, the emphasis has been placed upon succinctness and the use of symbolism wherever possible. As a result, a person uneducated in ALGOL cannot readily understand any ALGOL program. COBOL, on the other hand, has placed a major emphasis on readability, even to the point of being verbose. A person who has never seen a COBOL manual before can get a fairly good idea of what is going on from a normal COBOL program.

Again, as above, no attempt is being made to sit in judgment on these views, but merely to point them out.

(d) Data Description

The major difference within the area of data description is of course the fact that COBOL has an extremely elaborate system while that in ALGOL is fairly simple. This, of course, only reflects the difference in the primary type of source problem as delineated in (a). To be more specific, COBOL makes heavy use of the concept of a file, whereas ALGOL does not allow for this at all. Furthermore, COBOL requires the user to specify the exact format and placement of both his input and output data, whereas ALGOL only concerns itself with specifying the type of variable. This, in turn, is of course directly caused by the emphasis on the need for a good input-output system in COBOL object programs and an omission of this factor from ALGOL.

(e) Input-Output

To cover the subject of input-output most simply, one needs only say that at least one-third of COBOL is devoted to this problem, whereas it is ignored completely in ALGOL. This is again a reflection of the very basic point given in (b).
3. Types and Levels of Interchangeability

In order to combine these languages with a minimum of change to either, it is necessary to specify the types and levels of interchangeability which exist. There are four of these which are of significance:

(a) Transliteration of Basic Symbols
(b) Transliteration of Groups of Symbols
(c) Translation of Groups of Symbols
(d) Arbitrary Differences

It must be re-emphasized that all references to ALGOL symbols are to the reference language. Actually, the job of combining these languages would be far simpler if one of the hardware representations for ALGOL were chosen, since COBOL is essentially a hardware representation.

The general meaning of (a) - (d) is given here, and Section III contains specific instances of these types of interchangeability.

(a) Transliteration of Basic Symbols

This simply involves a 1-1 correspondence between some of the basic symbols as defined in Section 2 of the ALGOL 60 report, and the character set as defined in Chapter III, 1 of the COBOL report.

(b) Transliteration of Groups of Symbols

In this case, it is possible to establish a particular sequence of ALGOL basic symbols and a particular sequence of COBOL words, because of the fact that a COBOL word is not considered a basic symbol.

(c) Translation of Groups of Symbols

In this case, there is not a 1-1 correspondence, but it is possible to translate certain specific groups of ALGOL symbols into certain groups of COBOL symbols, and conversely.

(d) Arbitrary Differences

There are certain differences in the notation which can be easily taken care of by making certain restrictions on one or the other or both languages.

II. DETAILED LISTING OF INTERCHANGEABILITY

Throughout the discussion of interchangeability, only the reference format of ALGOL is used. (See ALGOL report, INTRODUCTION.) Furthermore, no use or consideration is made of optional words in COBOL. (See COBOL report, III, 2.2.3b.)

To save writing, an arrowhead will be used to indicate the direction of the interchangeability. Thus A\rightarrow C means that the specified COBOL characters can be replaced by the designated ALGOL characters. A double-headed arrow means that the replacement can take place both ways.

These lists are not guaranteed to be complete, and in some cases, there are some very subtle points about these correspondences which are not discussed.

1. Transliteration of Basic Symbols

The list (Figure 1) shows the basic symbols in each language which have a 1-1 correspondence with basic symbols in the other language. This correspondence involves the syntax as well as the symbol itself, i.e., they are used the same way. (See ALGOL report, 2, 2.1-2.3 and COBOL III, 1.1 and 1.2).

2. Transliteration of Groups of Symbols

Many of the basic symbols in ALGOL have the same syntactic meaning as full words (which are groups of symbols) in COBOL. The list (Figure 2) actually consists primarily of basic ALGOL symbols and corresponding COBOL words or groups of words. These are considered to be transliterations because a 1-1 correspondence exists between the groups and no translation is required.

3. Translation of Groups of Symbols

It is extremely difficult to list all of the cases in which a group of symbols in one language can be translated into another. In some cases, the correspondence is so complicated as to render a translation either difficult and/or impractical. As an illustration of this problem, consider the existence of the standard functions which are in ALGOL, but not in COBOL. These can be handled either as a subroutine (which does not allow the programmer to ever write the actual form) or by means of the DEFINE. In the latter case, it is rather a moot point as to whether or not it is possible to mechanically translate the following COBOL statements into sin (x):

\[
\text{DEFINE VERB sin AS COMPUTE Z =}
\]

(appropriate formula for computing sin (x) ) \text{ WITH FORMAT sin (x) = Z.}
In order to computer 

\[ A = \frac{1}{2} \sin (Y) \]

one would have to write

\[ \text{SIN} (Y) = W \]

\[ \text{MULTIPLY} \ W \ \text{BY} \ 0.5 \ \text{GIVING} \ A \]

This point is mentioned again in Section III below.

With the realization that not all "translatable" groups can be shown, it is still possible to give some widely differing illustrations of cases where this is possible and practical. This is done in Figure 3.

4. **Arbitrary Differences**

The question of whether or not the difference between two expressions is "arbitrary" is certainly a matter of opinion and calls for a specific value judgment. Rather than stir up a storm on this minor issue, it seems better to say simply that any case in which a straight two-way transliteration is possible is considered arbitrary. Under this very restrictive definition, it should be noted that the only "arbitrary difference" is in the use of the symbols, \* and \( \div \) for multiplication in COBOL and ALGOL respectively, and the use of \( \div \) as a division operator in ALGOL.

III. **DEVIATIONS OF COBOL FROM ALGOL**

This section concerns itself with the ways in which COBOL deviates from ALGOL, considered only for the areas in which they basically coincide or overlap. (This same procedure is reversed in the next section, where COBOL is considered as the base.) Thus, the main discussion centers around the PROCEDURE DIVISION, with some comments on, and additions to, the DATA DIVISION.

The material shown here involves comparisons as well as associated suggestions and comments for changes which are needed to make COBOL contain equivalent capability to ALGOL. The general approach taken in both this section and the next is to add the necessary features to each language - without destroying its inherent characteristics - rather than placing restrictions. Thus, additions are made to each language within its own framework, so that equivalent capability is available although in a somewhat different form.

In order to avoid repeating material, both sections III and IV must be considered simultaneously if a complete comparison is desired.

1. **CHARACTERS AND WORDS**

a) **Word Formation**

COBOL differs from ALGOL by allowing purely numeric procedure-names, by allowing the letter in data-names to appear anywhere, and by restricting all words except literals to being less than or equal to 30 characters. The first two of these items (i.e., data and procedure-name differences) are somewhat arbitrary although there are good reasons for the choice made in each language. The restriction on the maximum number of characters for words is desirable for implementation reasons, although the number 30 is somewhat arbitrary.

One minor difference is that in COBOL, two procedure-names are equal only if they are identical, whereas in ALGOL, leading zeroes are ignored.

COBOL does not have any means of representing numbers in their floating point form. The ALGOL form cannot be adopted because of the lack of a suitable character for the "10". However, it is easy to add the concept in the DATA DIVISION.

b) **Subscripts**

Subscripting in COBOL is much more restricted than in ALGOL. From the point of view of the language, it is very easy to allow the more general form. Thus, to have equivalents in the two languages, COBOL subscripting must first be extended to allow any number; i.e., remove the restrictions to 3 subscripts. Then COBOL must allow subscripts to be subscripted themselves, and finally, must allow any arithmetic expression to be used as a subscript.

c) **Functions**

The ability to define arbitrary functions and the standard functions should both be added to COBOL. Note however, that it is not actually necessary to include this capability directly in the specifications because the same logical results can be achieved by using the DEFINE. The only disadvantage to this is the awkwardness of the method. (See II.3 above.)

2. **PROCEDURE DIVISION**

a) **General Procedural Syntax**

COBOL currently allows only 2 hierarchies of named procedures; i.e., paragraphs and sections. To cope with the more general ALGOL form, it is necessary to allow any number of nested named procedures in COBOL. This of course should be done by adding BEGIN and END with the same definitions as in ALGOL. Note that COBOL already contains "sentences" which are the logical equivalent of ALGOL compound statements. It is not necessary to introduce declarations into the PROCEDURE DIVISION since
the equivalent information can be imbedded into
the DATA DIVISION.

Since COBOL conditional statements
and sentences* contain the ALGOL form as a
special case, no change is needed. Furthermore,
the basic verb structure can be kept; since it
is shown below that ALGOL operations can be
expressed in terms of COBOL verbs.

ALGOL procedures can be handled
equivalently through the use of the DEFINE.

b) Arithmetic & MOVE Verbs

The five COBOL arithmetic verbs,
ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE,
as well as the MOVE, can be used to express
ALGOL assignment statements. It is necessary
to extend the capability of each verb to handle
floating point numbers.

c) Procedure Branching

The simple GO (i.e., Option 1)
of COBOL is of course equivalent to the go to of
ALGOL. The switch declaration in ALGOL can be
handled by the proper combination of GO TO ... DEPENDING ON, ALTER, and computational pro-
cedures.

The COBOL PERFORM is essentially a
special case of the ALGOL for. However, the
PERFORM must have several options added to it to
give it the full power and generality of for.
In particular, in the TIMES option, the ability
to show any number of these in a single PERFORM
must be added. The step until can be handled
provided the VARYING option of the PERFORM is
allowed to apply to any field and several
VARYING's can be written in one PERFORM. The
UNTIL and while are simply logical negations of
each other. Thus, the PERFORM can be extended
easily to make it equivalent to the ALGOL for.

The NOTE of COBOL and the comment
of ALGOL are equivalent. Furthermore, the use
of an extra semicolon to show a dummy statement
in ALGOL is equivalent to EXIT in COBOL.

3. DATA DIVISION

The major additions to the COBOL DATA
DIVISION are to the CLASS clause in the Record
Description. The additions will then result in
the equivalent of the type declarations. Allow-
ing BOOLEAN to be specified in COBOL will cause
the meaning to be identical in the two languages.
Any data-name which is NUMERIC and does not have
a decimal point is an ALGOL integer; all other
NUMERIC fields are real in the ALGOL sense.
Finally, a FLOATING-POINT category must be added.

* Defined in TC 68.1 - See Reference 2.

It should be noted that the informa-
tion given in an Array Declaration is basically
available in the OCCURS clause and the level
structure of the Record Description.

IV. DEVIATIONS OF ALGOL FROM COBOL

This section is similar to III in that it
deals with deviations of the two languages.
Therefore, the general remarks at the beginning
of III apply fairly well here, except of course
that the changes will be suggested for ALGOL to
give it equivalent capability to COBOL in the
areas where they are inherently similar. This
latter qualification is extremely important,
because no attempt is being made to add full
description of data, input-output, or environ-
ment to ALGOL.

In order to avoid repeating material, both
sections III and IV must be considered simul-
taneously if a complete comparison is desired.

1. BASIC SYMBOLS AND CONCEPTS

a) Letters

The use of both upper and lower case
letters cannot be handled in COBOL for hardware
reasons. Therefore, the use of the lower case
letters in ALGOL should not be allowed.

b) Identifiers

The requirement for a letter at the
beginning of each identifier should be removed.
Furthermore, labels can be allowed to be purely
numeric since they can be identified from the
context. A restriction on the length of any
identifier to 30 characters seems reasonable
when implementation is being considered.

c) Variable Types

Although ALGOL does not have all the
variable types that COBOL does (e.g. ALPHABETIC)
it does not seem meaningful to add this ability
since there is no real way to handle it, and
adding this to ALGOL is inherently dependent on
large extensions in the area of data description.

2. SYNTAX

a) Conditional Statements

The primary addition to ALGOL needed
to cover the COBOL syntax is the extension of
conditional statements to allow nested "if
clauses". This form is one of several pro-
posals that has been made to handle the ambig-
ity which now exists in ALGOL. (See 4.5 in
ALGOL report.)

b) for Statements

The main change needed here is to
add the ability to apply the for to procedures
other than those immediately following it.
3. COBOL VERBS

Because of the importance of the verbs in COBOL, it is desirable to discuss explicitly their ALGOL counterparts.

a) Arithmetic and Data Movement

Some of the COBOL verbs appear on the surface to have no ALGOL equivalent, but this is not really true. As pointed out in III, the five arithmetic verbs and the MOVE are equivalent to assignment statements. The EXAMINE which appears to have no direct counterpart in ALGOL actually does not need one, since the EXAMINE is not primitive. That is, the EXAMINE can always be expressed as a proper combination of PERFORM, equality tests, and MOVE. Since these can always be put into correspondence with ALGOL concepts, the EXAMINE is then just a summation of these correspondences.

b) Procedure Branching Verbs

See III, 2(c) above.

c) STOP Verb

The ability to signify the stopping point of a program does not appear in ALGOL. This can be handled either by modifying the end or by just adding a separator STOP.

d) Input-Output Verbs

There is obviously no ALGOL equivalent for this.

e) Compiler Directing Verbs

The ENTER is of course equivalent to a procedure so there is no problem here. The EXIT is equivalent to the use of an extra semicolon in ALGOL. The USE and INCLUDE have no direct counterpart in ALGOL since there is no input-output of any kind, nor any reference to a library. However, the same effect can be obtained through the proper use of procedures. The DEFINE is essentially equivalent to a procedure. Finally, the NOTE is, of course, equivalent to comment.

4. DATA AND ENVIRONMENT DESCRIPTION

For the reasons given in I.1 above, these COBOL elements are not existent in ALGOL, except for a few special cases which are handled by type declarations. Files as such cannot be described in ALGOL. Similarly, ALGOL contains no provision for describing any of the environmental conditions under which the problem is to be run on a computer. It is beyond the scope of this paper to consider this type of addition to ALGOL.

V. COMMENTS AND CONCLUSIONS

This paper has chosen a method whereby ALGOL and COBOL (as defined in the references) can be combined to produce a new language called ALABOL, which will contain major portions of ALGOL and COBOL as subsets. This work has been done under the strict - although self-imposed - requirement that neither language should have its basic structure altered in order to conform to the other. Because this was the basic approach taken, the resultant combination is perhaps less powerful than might otherwise be achieved. On the other hand, it is a very practical fact of life that the circumstances under which both of these languages were created do not lend themselves kindly to major conceptual changes. It is the strong opinion of the author that the additions (and changes where necessary) are definitely not major. It seems that if both the maintenance committees really wish to take advantage of the power to be gained by this method, then this can be done fairly easily. None of the changes or additions appear to cause any large amount of difficulty; in fact, some of them have already been discussed as "something to be done in the future".

As was stated earlier, this paper definitely does not intend to present the final specifications for a new language. What it has done is point out quite specifically just what changes and additions are needed to bring the languages together under the ground rules previously discussed. In most cases, full details have been given, whereas in some other instances, some of the details are available but not shown in the paper. It should also be noted that this paper did not include the addition to ALGOL of input-output, full data description, or environmental descriptions to ALGOL. This would have involved a very major conceptual change in the structure of the language and, therefore, would violate one of the current ground rules. It is expected that this will be done in the future.

The method shown here is practical and not difficult to apply. It should be of interest to users and manufacturers, to both members and non-members of the maintenance committees, and to all members of the computing industry who have any interest in the development of powerful procedural and problem-oriented languages.

REFERENCES


2. COBOL: Report to Conference on Data Systems Languages, April 1960, U.S. Government Printing Office #1960 0-552133 plus 3 supplements distributed to CODASYL mailing list, plus change labeled TC 68.1 which has been officially approved by the maintenance committees.
<table>
<thead>
<tr>
<th>ALOOL</th>
<th>COBOL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1,...,9</td>
<td>0,1,...,9</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>A,B,C,...,Z</td>
<td>A,B,C,...,Z</td>
<td>Only upper case letters are allowed in COBOL.</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>Used to indicate subtraction. Since this character has two uses in COBOL, the arrow cannot go both ways.</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&lt;=</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>&gt;=</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>Identical symbols.</td>
</tr>
<tr>
<td></td>
<td>Space</td>
<td># is used only in ALOOL strings and so the arrow can go only one way.</td>
</tr>
<tr>
<td>[</td>
<td>(</td>
<td>Used as subscript delimiters. The use of parentheses for arithmetic expressions in COBOL prevents the arrow from going both ways.</td>
</tr>
<tr>
<td>]</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>(</td>
<td>Used in arithmetic expressions.</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>The pair '&quot;' serve as string delimiters in ALOOL which are similar conceptually to literals in COBOL. Because ALOOL uses two symbols, there is not a full double correspondence.</td>
</tr>
<tr>
<td>end</td>
<td>.</td>
<td>The use of the decimal point for numbers in COBOL prevents the double correspondence.</td>
</tr>
</tbody>
</table>

Figure 1
<table>
<thead>
<tr>
<th>ALGOL</th>
<th>COBOL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>; =</td>
<td>EQUALS</td>
<td>ALGOL basic symbol, COBOL word</td>
</tr>
<tr>
<td>; =</td>
<td>EQUAL TO</td>
<td>ALGOL basic symbol, group of COBOL words</td>
</tr>
<tr>
<td>\geq 0</td>
<td>POSITIVE OR ZERO</td>
<td>Groups of symbols in both languages</td>
</tr>
<tr>
<td>\leq 0</td>
<td>NEGATIVE OR ZERO</td>
<td>Groups of symbols in both languages</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>POSITIVE</td>
<td>ALGOL group of symbols, COBOL word</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>NEGATIVE</td>
<td>ALGOL group of symbols, COBOL word</td>
</tr>
<tr>
<td>&gt;</td>
<td>IS GREATER THAN</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>EXCEEDS</td>
<td>ALGOL basic symbol, group of COBOL symbols</td>
</tr>
<tr>
<td>\up</td>
<td>#</td>
<td>ALGOL basic symbol, group of COBOL symbols</td>
</tr>
<tr>
<td>A[X,Y]</td>
<td>A(X,Y)</td>
<td>Groups of symbols in both languages</td>
</tr>
<tr>
<td>comment</td>
<td>NOTE</td>
<td>ALGOL basic symbol, COBOL word</td>
</tr>
<tr>
<td>go to</td>
<td>GO</td>
<td>ALGOL basic symbol, COBOL word</td>
</tr>
<tr>
<td>\lor</td>
<td>OR</td>
<td>ALGOL basic symbol, COBOL word</td>
</tr>
<tr>
<td>\land</td>
<td>AND</td>
<td>ALGOL basic symbol, COBOL word</td>
</tr>
<tr>
<td>\lnot</td>
<td>NOT</td>
<td>ALGOL basic symbol, COBOL word</td>
</tr>
</tbody>
</table>

**Figure 2**

<table>
<thead>
<tr>
<th>ALGOL</th>
<th>COBOL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,2 \times 10^2</td>
<td>720</td>
<td>This is just one example of the whole class of ALGOL representations of numbers written in exponential form.</td>
</tr>
<tr>
<td>Y_i = X-5</td>
<td>COMPUTE Y = X-5</td>
<td>This is, of course, just an illustration since the COMPUTE verb is equivalent to the # in ALGOL.</td>
</tr>
<tr>
<td>for x = 1 step until n \times PERFORM ... n TIMES</td>
<td></td>
<td>This is again an illustration.</td>
</tr>
</tbody>
</table>

**Figure 3**