R.M. De Morgan, I.D. Hill, B.A. Wichmann

A draft of this document was produced for a meeting of the IFIP Working Group 2.1 held in Breukelen, August 1974. Changes have been made as a result of comments received at that meeting.

The authors would like comments on whether the primitive IFIP based input-output system is worth including in this document. Comments would also be welcome on 5.2.4.3 which permits the declaration of arrays containing no element.

The authors have failed to reach agreement on whether subscripted controlled variables should continue to be allowed, or whether a restriction should be made (as in the IFIP subset) to allow only a variable identifier to be a controlled variable.

For the present this commentary has been written to make the restriction, although under 4.6.4.2 an explanation is given of how the operations on a subscripted controlled variable should be defined if allowed. If it is to be allowed, various consequential changes would be needed elsewhere in the document.

Comments on this issue would be welcomed.

Would AB readers please send comments to:

B. A. Wichmann, National Physical Laboratory,
Teddington, Middlesex, TW11 0LW U.K.

Owing to the limitations of the ISO-code printing device, the following representations are used:

space	.
string quotes	( )
or	or
and	and
not	not
implies	impl
equivalent	equiv
not equals	ne
integer divide	div
ten	&
multiplication	*

also syntactic brackets are not distinguished from less than and greater than.

A commentary on the ALGOL 60 Revised Report

R.M. De Morgan, I.D. Hill, B.A.Wichmann

"For, as on the one side common experience sheweth, that where a change hath been made of things advisedly established (no evident necessity so requiring) sundry inconveniences have thereupon ensued; and those many times more and greater than the evils, that were intended to be remedied by such change: So on the other side, the particular Forms .... being things in their own nature indifferent, and alterable, and so acknowledged; it is but reasonable, that upon weighty and important considerations, according to the various exigency of times and occasions, such changes and alterations should be made therein, as to those that are in place of Authority should from time to time seem either necessary or expedient ....

And therefore of the sundry alterations proposed unto us, we have rejected all such as were either of dangerous consequence .... or else of no consequence at all, but utterly frivolous and vain ....

Our general aim therefore in this undertaking was, not to gratify this or that party in any their unreasonable demands; but to do that, which to our best understandings we conceived might most tend to the preservation of Peace and Unity ....

If any man, who shall desire a more particular account of the several Alterations .... shall take the pains to compare the present Book with the former; we doubt not but the reason of the change may easily appear."

Preface to Book of Common Prayer 1662.

Over the past eleven years, various defects have been noted in the 'Revised Report on the Algorithmic Language ALGOL 60'. In general, these defects are of little consequence, but have resulted in unnecessary variations in the various implementations of ALGOL 60 thus impairing the portability of ALGOL 60 algorithms. The body responsible for ALGOL 60, Working Group 2.1 of the International Federation for Information Processing, therefore asked a small group under the chairmanship of C.A.R. Hoare to examine the maintenance of ALGOL 60. As a result of an appeal by Professor Hoare, about a dozen letters were received expressing views on the work that should be undertaken. Unfortunately, the views were often conflicting so it has not been possible to satisfy them all.

Although ALGOL 60 shows signs of being swamped by the expanding use of FORTRAN, and although ALGOL 68 exists, the remaining usage of the language is still significant and it remains much loved by its users.

The constancy of the language over many years should be regarded as one of its assets, not lightly to be disturbed. Changes should be kept to the minimum of necessary clarifications. Any large extensions, at this stage, would be doomed to be ignored, whereas we hope that the relatively small changes that we are suggesting may be incorporated into existing compilers.

It would seem wrong, after the Revised Report has existed unchanged for so many years, to try to force any changes by, for example, withdrawing IFIP recognition from the 1962 version in favour of any new proposals.

The suggestion, therefore, is that these proposals should be taken as defining a new language, to be called ALGOL 60.1, which, at least for awhile, would exist in parallel with Revised ALGOL 60, and reactions would be evaluated before reaching any final conclusion.

Two items that we have rejected, as being a little too radical, but that we should regard as strong candidates for consideration if it were decided to be bolder are (i) the iterative statement: while <Boolean expression> do <statement> (ii) the conditional string, defined by:

<simple string> ::= (<open string>) | (<string>)
<string> ::= <simple string>|<if clause><simple string>else<string>

We believe that there would be general (though not quite universal) rejoicing among ALGOL devotees if the extended input-output procedures of Knuth et al. (1964), and of ISO/R 1538 Part II B, were to be repudiated. In our commentary we have simply ignored them for the present.

We have not attempted to change the structure of the subsets, as defined in the ISO Recommendation, but in some instances (as detailed below) we believe that the present subset restrictions should apply to the full language (level 0). Also, having only six significant characters in an identifier at level 1 (ECMA subset with recursion) we feel is unduly restrictive. At levels 2 and 3 (the ECMA and IFIP subsets), it may be more difficult to ensure adherence to the additional restrictions than compile the full language.

This paper is in the form of a commentary on the Revised Report although most of these comments are expressed in the form of amendments. A booklet containing this paper, the Revised Report and our amendments applied to the Revised Report will be available[9].

A summary of our suggestions for language modification (as distinct from changes of wording without any change of intention) is as follows:

1. own variables are to be regarded as static. own arrays may only have fixed bounds. All own variables are to be initialised to zero or false.

2. The for statement is to be dynamic, but a step expression will be evaluated only once each time around the loop. The controlled variable cannot be a subscripted variable.

3. The controlled variable of a for statement will remain defined after exit from the loop.

4. Comments and strings are to consist of characters, not of ALGOL basic symbols, the characters allowed being implementation dependent.

5. Some new standard functions and procedures are introduced, including environmental enquiries and elementary transput.

6. Numerical labels are abandoned.

7. The effect of a go to statement leading to an undefined switch designator is to become undefined.

8. All formal parameters must be specified.

9. The exponentiation operator is to become undefined if both operands are of integer type, and the exponent is negative.

Introduction

The Revised Report explicitly notes in the Introduction that five issues have been left unresolved and await further clarification. Our views on these matters are as follows:-

Side effects of functions

Side effects of functions should be permitted without restriction, since it does not seem feasible to outlaw foolish uses without at the same time outlawing sensible uses. It is the programmer's responsibility not to employ the foolish uses.

It should be noted, in particular, that the Revised Report does not always specify the order in which expressions, or primaries within an expression, are to be evaluated. For instance, 3.3.5 specifies the order of execution of operations, but leaves undefined the order of evaluation of the primaries for those operations.

If different permitted orders of evaluation will produce different results, due to the action of side effects, then the action of the program must be regarded as undefined, in the sense of the footnote to the Revised Report, section 1. It should be noted that in the evaluation of a simple expression (either Boolean or arithmetic) all the primaries of the expression must be evaluated unless a jump out of a function is taken. A primary may contain expressions. The evaluation of a primary does not necessarily require the evaluation of all such expressions.

The 'call by name' concept

There appears to be a need to modify to only a minor extent the detailed description of the execution of a procedure statement in 4.7. The exact effect of the call-by-name mechanism is there defined. See the commentary on 4.7.3.2 for the detailed amendment.

Own: static or dynamic

The static interpretation of own is now accepted as standard. Ehat is to say: an own variable behaves exactly as if it had been declared in a block head immediately preceding the program, except that it is accessible only within its own scope. An extra end, corresponding to this fictitious block head, is assumed to follow the final end of the program. Possible conflicts between identifiers, resulting from this process, are resolved by suitable systematic changes of the identifiers involved.

It follows that: (i) an own variable, declared in a block within a procedure, which is called from different parts of the program, represents the same variable every time, not a separate variable for each place of call; (ii) an own variable, declared within a procedure that is activated recursively, represents the same variable at every level of the recursion; (iii) if a complete program is labelled, a go to leading to this label does not affect the values of own variables.

Furthermore, we recommend that this fictitious block should serve not only to declare any own variables, but also to assign initial values to them. All integer and real own variables should be assigned the value 0, while all Boolean own variables should be assigned the value false.

The bounds of an own array must be of the form <integer>. The second example of 5.2.2 must therefore be regarded as incorrect.

For statement: static or dynamic

The dynamic interpretation of the for statement has become accepted as standard, to such an extent that to many ALGOL 60 users it comes as a severe shock to be told that the Revised Report does not specify that this is the required interpretation. Having accepted the dynamic version, however, it still needs to be settled whether the step-expression has to be evaluated more than once per cycle, when a step-until element is being executed. The exact meaning of a subscripted controlled variable is also a matter of difficulty. It is now to be regarded as standard that the step expression should be evaluated once only per cycle, and that subscripted controlled variables should be forbidden. See the commentary on 4.6 below for the detailed amendments.

Conflict between specification and declaration

The Revised Report section 4.7.5 requires that the kind and type of each actual parameter be compatible with the kind and type of the corresponding formal parameter. This compatibility is defined by means of a table which appears under the commentary on that section.

In addition, the Introduction recognizes three different levels of language, Reference, Publication and Hardware. We propose that these should be reduced to Reference and Hardware only.

Publication language

The concept of publication language should no longer be recognised. It has become the universal practice that ALGOL 60 publications use reference language, with occasional minor variations in representation. These variations however (such as and for $\wedge$, or * for $\times$) are rarely, if ever, those recommended in the Revised Report for publication language.

Furthermore the wording of the Revised Report does not agree with what was presumably the intention, since removal of the upward arrow, as well as raising the exponent, was surely intended for exponentiation.

There is also an ambiguity introduced, since in reference language 2&5

is a number of real type, whereas 2*10↑5 is an expression of integer type. Yet both become $2*10^5$ in publication language.

1 Structure of the language

The environmental block

A program is always considered to be contained within an additional level of block structure. This block is called the environmental block, and contains declarations of standard functions, input and output procedures, and possibly other procedures to be made available without declaration within the program as well as the fictitious declaration of <u>own</u> variables.

The environmental block includes declarations of at least the following procedures:

abs, iabs, sign, entier,
sqrt, sin, cos, arctan, ln, exp,
maxreal, minreal, maxint, epsilon,
fault, stop,
insymbol, outsymbol, inreal, outreal, ininteger,
outterminator, outinteger, outstring, length.

It should be noted that since the environmental block is simply an ALGOL block, these identifiers may be redeclared within any other block if desired, with the usual scope rules applying.

The penultimate paragraph of section 1 should be amended to read:

'A program is a block or a compound statement that is contained only within a fictitious block, always assumed to be present, called the environmental block, and that makes no use of statements not contained within itself, except that it may invoke such procedure identifiers and function designators as may be assumed to be declared in the environmental block.

The environmental block contains procedure declarations of standard functions, input and output operations, and possibly other operations to be made available without declaration within the program. It also contains the fictitious declaration, and initialisation, of <u>own</u> variables (see section 5).'

The fictitious structure surrounding the program is:

```
begin
 <declaration of standard functions and procedures>;
 <fictitious declaration of own variables>;
 <initialisation of own variables>;
 <program>;
 Ω :
end
```

where Ω is a label that is not accessible within the program but may be used by standard functions or procedures. Note that with this amendment the program 'sin: <u>begin</u> <u>end</u>' is no longer valid.

2.3 Delimiters

Footnote concerning do

The footnote to 2.3, and the symbol that refers to this footnote (at the end of the definition of <sequential operator>), should both be deleted. It is unnecessary and confusing to readers who have no knowledge of the preliminary report, and also causes unnecessary ambiguity in the interpretation of the metalinguistic formulae. How can one tell that 'do "' (in the Comp.J. version), 'do[7]' (in the Comm. ACM. version), 'do[2]' (in the Num. Math. version), or 'do *' (in the ISO version) is not the required basic symbol?

Space symbol

In line with the other modifications concerning strings (see 2.6), there is now no need for the space symbol in the Reference Language. Hence ␣ can now be deleted from the list of separators in 2.3. However, it is recommended that a visible character is used to represent a space so that typographical features are ignored throughout the language.

Characters in comments

Section 2.3 allows only basic symbols within comments, although most compilers allow any hardware character and published ALGOL 60 often allows anything except semicolon. Indeed, the Revised Report examples contain several additional characters.

The relevant part of 2.3 should now read:

'The sequence	is equivalent to
;comment <any sequence of zero or more characters not containing ;>;	;
begin comment <any sequence of zero or more characters not containing ;>;	begin
end <any sequence of zero or more basic symbols not containing end or else or ;>	end

This permits any characters after comment. It should be noted that the third type of comment (following end) is still restricted, since seeking for end or ; or else is more difficult for a compiler than merely seeking for ;.'

2.6.1 Syntax

ALGOL 60 is not, and is not intended to be, a string manipulation language. The only use of strings is in communication to and from foreign media. It must be recognised that such foreign media deal in characters, not in ALGOL basic symbols. To be useful, the concept of a string must be put in touch with reality and be defined in terms of characters.

Characters are already recognised as existing in section 2.1 which says that the 'alphabet may ... be ... extended with any other distinctive character'. What characters are available must be a matter of hardware representation and be left undefined by the reference language just as 'code' is (see 5.4.6), except in insisting that string quotes must match, so that the end of a string can be detected.

To conform with the suggested change in strings to a sequence of characters and also to clarify the definition of <open string>, the syntax now becomes:-

<proper string> ::= <any sequence of characters not containing
(or) >|<empty>
<open string> ::= <proper string>|<open string><string><proper string>

2.6.2 Examples

The character ., which is not now a basic symbol, is used to represent the position in a string at which a space is required.

2.6.3 Semantics

This section should now read:-

'In order to enable the language to handle sequences of characters the string quotes (and) are introduced.

The characters available within a string are a question of hardware representation, and further rules are not given in the reference language. However it is recommended that, in strings as elsewhere, typographical features such as blank space or change to a new line should have no significance, and that the character . should be used to represent a space.

Strings are used as actual parameters of procedures (see Sections 3.2 Function designators and 4.7 Procedure statements).'

3 Expressions

In the introduction to this section, the list of constituents of expressions omitted labels and switch designators. The second sentence should therefore read: 'Constituents of these expressions, except for certain delimiters, are logical values, numbers, variables, function designators, labels, switch designators, and elementary arithmetic, relational, logical, and sequential operators.'

3.1.3 Semantics

Add to this section:

'The value of a variable, not declared <u>own</u>, is undefined from entry into the block in which it is declared until an assignment is made to it.'

This brings variables into line with function values (see 5.4.4).

3.2.4 Standard functions

Replace the existing sections 3.2.4 and 3.2.5 by

'3.2.4 Standard functions and procedures

Certain standard functions and procedures are declared in the environmental block with the following procedure identifiers:
abs, iabs, sign, entier, sqrt, sin, cos, arctan, ln, exp, insymbol, outsymbol, length, outstring, outterminator, stop, fault, ininteger, outinteger, inreal, outreal, maxreal, minreal, maxint, and epsilon.

For details of these functions and procedures, see the specification of the environmental block given as Example 3, at the end of the report.'

The identifiers maxreal, minreal, maxint, and epsilon define functions, not standard variables, partly to avoid introducing a new concept unnecessarily, but mainly so as to make it impossible to assign to them.

3.2.5 Transfer functions

As with the other standard functions 'entier' must be provided in the environmental block and is not just a recommendation.

Section 3.2.5 should be deleted, since its purpose is now included in the new version of 3.2.4 given above.

3.3 Arithmetic expressions

3.3.3 Semantics

The largest arithmetic expression

The word 'longest' should be substituted for 'largest' in '(the largest arithmetic expression found in this position is understood)', since 'largest' might be taken as referring to the value of the expression.

Meaning of else

The final sentence of this section should be deleted. It is incorrect since

else <simple arithmetic expression>

must not be followed by a further else, whereas

else if true then <simple arithmetic expression>

must be followed by a further else. The two constructions are therefore not equivalent.

It should be replaced by
'If none of the Boolean expressions has the value true, then the value of the arithmetic expression is the value of the expression following the final else'.

3.3.4.2 Division operators

Amend the first sentence by changing 'denote division, to be understood' to read 'denote division. The operations are undefined if the factor has the value zero, but are otherwise to be understood'.

It should be noted that the word 'mathematically', in the definition of integer division, is intended to signify that the specified operations are to be performed without rounding error.

The result of integer division can be given by means of a function. Hence the words 'mathematically defined as follows:' to the end of the section should be replaced by 'if a and b are of integer type, then the value of a div b is given by the function:

```
integer procedure div(a, b); value a, b;
     integer a, b;
     if b = 0 then
            fault( (div.by.zero) , a)
     else
            begin integer q, r;
            q := 0;  r := iabs(a);
            for r := r - iabs(b) while r ≥ 0 do
                 q := q + 1;
            div := if a < 0 equiv b > 0 then -q else q
            end div'
```

It should be noted that although real expressions could be used as arguments to the procedure div, the operator div is permitted only with operands of type integer. It also should be noted that div is not a standard function.

3.3.4.3 Exponentiation operator

Rather than give a table of values given by this operator, it seems more appropriate to define the values by means of algorithms. To achieve this, the second half of this section starting 'Writing i for a number ...' can be replaced by :-

'If r is of real type and x of either real or integer type, then the value of x↑r is given by the function:

```
real procedure expr(x, r); value x, r;
      real x, r;
      if x > 0.0 then
           expr := exp(r*ln(x))
      else if x = 0.0 and r > 0.0 then
           expr := 0.0
      else
           fault( (expr_undefined) , x)
```

If n is of integer type and x of real type, then the value of x↑n is given by the function:

```
real procedure expn(x, n); value x, n;
      real x;  integer n;
      if n = 0 and x = 0.0 then
           fault( (0.0↑0) , x)
      else
           begin
           real result;  integer i;
           result := 1.0;
           for i := iabs(n) step -1 until 1 do
                 result := result*x;
           expn := if n<0 then 1.0/result else result
           end expn
```

If i and j are both of integer type, then the value of i↑j is given by the function:

```
integer procedure expi(i, j);  value i, j;
        integer i, j;
        if j < 0 or i = 0 and j = 0 then
              fault( (expi_undefined), j)
        else
              begin
              integer k, result;
              result := 1;
              for k := 1 step 1 until j do
                   result := result * i;
              expi := result
              end expi
```

The call of the procedure fault denotes that the action of the program is undefined. The numerical accuracy of particular implementations of this operator should be no worse than that produced by the above algorithms.'

The Revised Report contains a difficulty with this operator in that the type of <integer>↑<integer> depends upon the sign of the exponent. The above implementation is undefined if the factor and primary are of type integer and the primary is negative. If it is desired that a real result should be produced then i↑j can be written as float(i)↑j where float is a function which gives the real value as in the assignment float := i. It should be noted that float is not a standard function.

In many ways a much neater solution would be to have two different symbols, for real exponentiation and integer exponentiation, in a similar manner to real and integer division, but the above seems the best compromise, as we do not consider that it would be wise to introduce any new basic symbol.

3.3.4.4 Type of a conditional expression

Since the type of a conditional expression is not specified in the Revised Report, a new section is required thus:-

The type of an arithmetic expression of the form

    if B then SAE else AE

does not depend upon the value of B. The expression is of type real if either SAE or AE is real and is of type integer otherwise.

3.3.5 Precedence of operators

It should be noted that although the precedence of operators determines the order in which the operations are performed, the order of evaluation of the primaries for these operations is not defined.

3.3.6 Arithmetics of real quantities

The reference to 'hardware representations' should be replaced by 'implementations', since elsewhere in the Revised Report 'hardware representation' refers to the representation of basic symbols.

3.4 Boolean expressions

3.4.5 The operators

Insert as the first sentence 'The relational operators <, ≤, =, ≥, > and ne have their conventional meaning (less than, less than or equal to, equal to, greater than or equal to, greater than, not equal to).'

3.5 Designational expressions

3.5.1 Syntax

Numerical labels

Numerical labels add in no way to the power or usefulness of the language although providing difficulties for the compiler-writer. They must now be regarded as obsolete in the full language as well as in the subsets. The syntax should now be

<label> ::= <identifier>

3.5.2 Examples

To conform to the change in labels, in the first and last examples, replace 17 by L17.

3.5.5 Unsigned integers as labels

Delete this section.

4 Statements

4.1 Compound statements and blocks

4.1.3 Semantics

Replace the last sentence of the second paragraph by:

'A label is said to be implicitly declared in this block head, as distinct from the explicit declaration of all other local identifiers. In this context a procedure body, or the statement following a <u>for</u> clause, must be considered as if it were enclosed by <u>begin</u> and <u>end</u> and treated as a block. A label that is not within any block of the program (nor within a procedure body, or the statement following a <u>for</u> clause) is implicitly declared in the head of the environmental block.'

4.2 Assignment statements

4.2.3 Semantics

Amend 'the body of a procedure defining the value of a function designator' to read 'the body of the procedure defining the value of the function designator denoted by that identifier.' This ensures that an assignment to a function can occur only within that function.

To conform to the requirement on access to a subscripted variable add to this section:

'If assignment is made to a subscripted variable, the values of all the subscripts must lie within the appropriate subscript bounds. Otherwise the action of the program becomes undefined.'

4.2.4 Types

Replace the wording 'equivalent to entier (E + 0.5)' by 'which is the largest integral quantity not exceeding E + 0.5 in the mathematical sense (i.e. without rounding error).'

4.3 Go to statements

4.3.2 Examples

The labels 8 and 17 be must replaced by L8 and L17 respectively since integer labels are no longer permitted.

4.3.5 Go to an undefined switch designator

Replace this section by:

'A go to statement is undefined if the designational expression is a switch designator whose value is undefined.'

4.4 Dummy statements

4.4.2 Examples

Amend the second example to read

<u>begin</u> statements; John: <u>end</u>

This is necessary since '...' is not valid ALGOL 60.

4.5 Conditional statements

4.5.3.1 If statement

Reword this section as follows:

'An <u>if</u> statement is of the form

<u>if</u> B <u>then</u> Su

where B is a Boolean expression and Su is an unconditional statement. In execution, B is evaluated; if the result is <u>true</u>, Su is executed; if the result is <u>false</u>, Su is not executed.

If Su contains a label, and a <u>go to</u> statement leads to the label, then B is not evaluated, and the computation continues with execution of the labelled statement.'

4.5.3.2 Conditional statement

Reword this section as follows:

'Three forms of unlabelled conditional statement exist, namely:

<u>if</u> B <u>then</u> Su
<u>if</u> B <u>then</u> Sfor
<u>if</u> B <u>then</u> Su <u>else</u> S

where Su is an unconditional statement, Sfor is a <u>for</u> statement and S is a statement.

The meaning of the first form is given in 4.5.3.1.

The second form is equivalent to

<u>if</u> B <u>then</u> <u>begin</u> Sfor <u>end</u>

The third form is equivalent to

```
     begin
     if B then begin Su; goto L4 end;
     S;
L4:  end
```

If the use of L4 causes any clash of identifiers it must be systematically changed to some other identifier - in particular, if S is conditional, and also of this form, a different label must be used in following the same rule.'

4.5.4 Go to into a conditional statement

Delete the last three words and substitute 'execution of a conditional statement.'

4.6 For statements

The exact interpretation of the ALGOL 60 for loop mechanism is controversial. The method given below has the advantage of being expressed in ALGOL 60.

4.6.1 Syntax

Replace the syntax of <for clause> by

```
<for clause> ::= for <variable identifier> := <for list> do
```

4.6.3 Semantics

Replace this section by:

'A for clause causes the statement S which it precedes to be repeatedly executed zero or more times. In addition it performs a sequence of assignments to its controlled variable (the variable after for). The controlled variable must be of real or integer type.'

4.6.4 The for list elements

Replace this section by:

'If the for list contains more than one element then

for V := X, Y do S where X is a for list element, and Y is a for list (which may consist of one element or more), is equivalent to

```
begin
procedure S1;  S;
for V := X do S1;
for V := Y do S1
end
```

Repeated use of this rule enables any for statement with n elements to be changed to n for statements with one element each. If the use of S1 causes any clash of identifiers it must be systematically changed to some other identifier.'

4.6.4.1 Arithmetic expression element

Replace this section by:

'If X is an arithmetic expression

```
     for V := X do S
```

is equivalent to

```
              begin
              V := X;  S
              end
```

where S is treated as if it were a block (see 4.1.3).'

4.6.4.2 Step-until element

Replace this section by:

'for V := A step B until C do S

is equivalent to

```
              begin <type of B> D;
              V := A;  D := B;
          L1: if (V-C)*sign(D) ≤ 0 then
                   begin
                   S;  V := V+D;
                   D := B;  goto L1
                   end
              end
```

where S is treated as if it were a block (see 4.1.3).

In the above, <type of B> must be replaced by real or integer according to the type of B. If the use of D, or of L1, causes any clash of identifiers, it must be systematically changed to some other identifier.'

If it were decided to allow subscripted controlled variables, the method should be:

```
     for V[i] := A step B until C do S
```

is to mean

```
     begin <type of B> D; integer j;
     j := i;  V[j] := A;  D := B;
 L1: if (V[j] - C) * sign(D) ≤ 0 then
            begin
            S; j := i;
            V[j] := V[j] + D;  D := B;
            goto L1
            end
     end
```

and similarly with controlled variables having more than one subscript.

4.6.4.3 While element

Replace this section by:

'for V := E while F do S

is equivalent to

```
        begin
   L3:  V := E;
        if F then
                begin
                S;  goto L3
                end
        end
```

where S is treated as if it were a block (see 4.1.3). If the use of L3 causes any clash of identifiers it must be systematically changed to some other identifier.'

4.6.5 The value of the controlled variable upon exit

Replace this section by:

'Upon exit from the for statement, either through a go to statement, or by exhaustion of the for list, the controlled variable retains the last value assigned to it.'

4.6.6 Go to leading into a for statement

Replace this section by:

'The statement following a for clause always acts like a block, whether it has the form of one or not. Consequently the scope of any label within this statement can never extend beyond the statement.'

In general the rules given above are merely a tidying operation, removing certain ambiguities and uncertainties. However, there are some minor changes in what is to be regarded as correct ALGOL 60, as follows:

(i) for v[i] := <for list> do becomes incorrect, since a subscripted controlled variable is not allowed;

(ii) for j := A[i] while j=0 do i := i+1; examine(j) becomes correct, since j is defined after the for statement;

(iii) for j := k, m, n do q[j] := j; i := j becomes correct. j has the value n after the for statement;

(iv)

```
begin switch m := a,b;
.................
.................
for ....... do
      begin
      ..............
      ..............
      goto m[i];
      ..............
 a:   ..............
 b:   ..............
      end
end
```

becomes incorrect, since the scope of a and b does not extend to the switch declaration. The switch should be declared after the second begin instead of after the first;

(v)

```
            .............
            .............
            for ........ do
                   begin
                    ...........
                    ...........
              m:    ...........
                    ...........
                   end;
              ...........
         m:   ...........
              ...........
```

becomes correct, since the scope of the inner m does not extend beyond the for statement;

(vi) If the controlled variable is a name parameter, then the rules for a procedure call (see 4.7.3.2) prohibit the actual parameter from being a subscripted variable. The check for this restriction need be performed only on initial entry to the loop and not every time round the loop;

4.7 Procedure statements

4.7.3.2 Name replacement (call by name)

In the first sentence replace 'wherever syntactically possible' by 'if it is an expression but not a variable'. This avoids the difficulty with the existing wording that if procedure A has a parameter, that is passed to procedure B, procedure B may be unable to assign to it,since it may have been syntactically possible within A to put parentheses around it.

4.7.5 Restrictions

Amend the second sentence of the second paragraph to read: 'Some important particular cases of this general rule, and some additional restrictions, are the following:'

4.7.5.4

Add to this section:
'A label may be called by value, even though variables of type label do not exist.'

This facility is necessary at level 3, to allow a switch designator to be used as the actual parameter.

Add to this section:
'The correspondence between actual and formal parameters should be in accordance with the following table:

FORMAL PARAMETER	MODE	VALID ACTUAL PARAMETERS LEVEL 0	LEVELS 1,2	LEVEL 3
integer	value	ae	ae	ae
	name	ae*	ie*	is
real	value	ae	ae	ae
	name	ae*	re*	rs
Boolean	value	be	be	be
	name	be*	be*	bs
label	value	de	de	l,sd
	name	de	de	l
integer array+	value	aa	ia	ia
	name	ia	ia	ia
real array+	value	aa	ra	ra
	name	ra	ra	ra
Boolean array+	value	ba	ba	ba
	name	ba	ba	ba
typeless procedure+	name	ap,bp,tp	tp	tp
integer procedure+	name	ap	ip	ip
real procedure+	name	ap	rp	rp
Boolean procedure+	name	bp	bp	bp
switch	name	sw	sw	sw
string	name	st	st	st

key:designational:d

arithmetic:	a	expression:	e
integer:	i	simple variable:	s
real:	r	array:	a
Boolean:	b	procedure:	p
typeless:	t		

label:	l
switch designator:	sd
switch:	sw
actual string or string identifier:	st

* Where an assignment is made to the formal parameter, either explicitly in the body of the procedure, or implicitly by means of a further procedure call in which such an assignment is made, the actual parameter must be a variable.
+ With an array parameter, the number of subscripts appearing in any of

its subscript lists must agree with those of the actual parameter. Similarly, the number, kind and type of the parameters of a formal procedure parameter must agree with the actual parameter.

In a procedure call, for each corresponding pair of actual and formal parameters, the actual parameter A must satisfy the rules in the above table, depending on the type and mode of the formal parameter F.

If A is itself a formal parameter, it must satisfy the rules above depending solely on its specification, irrespective of the nature of its own actual parameter. Thus, if type conversion (e.g. integer-to-real) is required by the parameter substitution, this process takes place independent of the type of the actual parameter substituted for the formal parameter which is itself the actual parameter in the parameter substitution under consideration.'

The following example should make this clear:

```
begin
        real x, y;
        procedure p(i); integer i;
              q(i);
        procedure q(z); real z;
              y := z;
        x := 6.2;
        p(x)
end
```

The statement 'y := z' requires the evaluation of the actual parameter 'i' in p. This in turn requires the evaluation of the actual parameter 'x' in the outer block. A type conversion (real to integer) is invoked, giving 'i' a value of 6, and a further conversion (integer to real), giving 'z' the value 6.0. Hence, y is assigned the value 6.0.

4.7.9 Standard procedures

The Revised Report did not contain any procedures to handle input-output. To rectify this, and to facilitate the handling of error conditions, ten standard procedures are defined below. With the exception of outterminator, fault and stop, all these procedures appear in the IFIP recommendations for input-output[5]. However the IFIP procedures inarray and outarray have not been implemented, since their effect can be achieved by means of the procedures inreal and outreal within suitable for statements. The new section, defining these procedures is:-

'Ten standard procedures are defined, which are declared in the environmental block in an identical manner to the standard functions. These procedures are:- insymbol, outsymbol, outstring, ininteger, inreal, outinteger, outreal, outterminator, fault and stop. The input-output procedures identify physical devices or files by means of channel numbers which appear as the first parameter. The method by which this identification is achieved is outside the scope of this report. Each channel is regarded as containing a sequence of characters, the basic method of accessing or assigning these characters being via the procedures insymbol and outsymbol.

The procedures inreal and outreal are converses of each other in the sense that a channel containing characters from successive calls of outreal can be re-input by the same number of calls of inreal, but some accuracy may be lost. The procedures ininteger and outinteger are also a pair, but no accuracy can be lost. The procedure outterminator is called at the end of each of the procedures outreal, outinteger and outstring. Its action is machine dependent but it must ensure separation between successive output of numeric data.

These additional procedures are given as examples to illustrate the environmental block at the end of this report.'

5 Declarations

Delete the last two sentences ('Apart from labels ... one block head') and substitute the following:

'Apart from labels, formal parameters of procedure declarations, and identifiers declared in the environmental block, each identifier appearing in a program must be explicitly declared within the program.

No identifier may be declared either explicitly or implicitly (see 4.1.3) more than once in any one block head.'

5.1 Type declarations and 5.2 Array declarations

The syntax of 5.2.1 allows array, to be understood (5.2.3.3) as meaning real array. Yet own real array must be written in full, the abbreviation own array being prohibited.

To allow own array the following amendments should be made.

In 5.1.1 delete the definition of <local or own type> and <type declaration> and substitute:

<type declaration> ::= <type><type list>|own<type><type list>

In 5.2.1 delete the definition of <array declaration> and substitute:

<array declarer> ::= array<array list>|<type>array<array list>
<array declaration> ::= <array declarer>|own<array declarer>

5.1.3 Semantics

Because of the restrictions imposed upon exponentiation at level 3, a real variable cannot always be replaced by an integer variable. There are also difficulties at all levels with procedure parameters and hence, at all levels, the second paragraph of this section should be omitted.

5.2.2 Examples

The second example should be deleted, as an own array may only have constant bounds.

5.2.4 Lower upper bound expressions

Problems arise through the scope of identifiers appearing in these expressions which we hope are clarified by the following changes.

Replace section 5.2.4.2 by:

'5.2.4.2 The expression cannot include any identifier that is declared, either explicitly or implicitly (see 4.1.3), in the same block head as the array in question. The bounds of an array declared as <u>own</u> may only be of the syntactic form integer (see 2.5.1).'

Section 5.2.4.3 specifies the conditions under which an array is defined. An undefined array, in the sense of this section, should not be regarded as a fault but merely as giving an array of zero elements. To ensure this interpretation, add to this section 'If any lower subscript bound is greater than the corresponding upper bound, the array has no elements.'

The array identifier may then be used (for example as an actual parameter, even if called by value), but any reference to an element of the array will be incorrect.

Thus:

```
begin array A[1:n]; integer i;
.........
.........
for i := 1 step 1 until n do
         operate( A[i] );
.........
.........
end
```

is valid even if n=0. The array will not exist, but neither will its elements be accessed.

5.2.5 The identity of subscripted variables

This section should be deleted. The second sentence is no longer relevant, whereas the meaning, if any, of the first sentence is unclear.

5.4.3 Semantics

Add to the end of this section:

'No identifier may appear more than once in any one formal parameter list, nor may a formal parameter list contain the procedure identifier of the same procedure heading.'

5.4.4 Values of function designators

Modify 'in a left part' (in each of two places) to read 'as a left part'. This is necessary as a function designator can appear in a subscript expression in a left part.

A difficulty arises with a <u>go to</u> leading out of a function designator since if this jump is executed, no value for the function is defined. To

clarify that such jumps are permitted, at the end of the section add the following words:
'If a go to statement within the procedure, or within any other procedure activated by it, leads to an exit from the procedure, other than through its end, then the execution, of all statements that have been started but not yet completed and which do not contain the label to which the go to statement leads, is abandoned. The values of all variables that still have significance remain as they were immediately before execution of the go to statement.

If a function designator is used as a procedure statement, then the resulting value is lost, but such a statement may be used, if desired, for the purpose of invoking side effects.'

Some examples of jumping out of a function are:

```
 (i)      j := 3;
          j := p(L);
          ........
     L:   ........
```

If the jump is taken, j will still have the value 3 when L is reached.

```
 (ii)  procedure q(k);
          value k; integer k;
          begin
          .........
          .........
          end q;
          .........
          .........
          q(p(L));
          .........
     L:   .........
```

If the jump is taken, none of the statements of q will be performed.

```
 (iii)  i := m[k] := n[p(L)] := s[t] := j := 3;
              ........
        L:    ........
```

If the jump is taken, none of the variables will have the value 3 assigned to it. Any side effects due to evaluation of k will have been performed; any side effects due to evaluation of t will not (see 4.2.3.1, 4.2.3.2 and 4.2.3.3).

```
 (iv)       L: ........
               ........
            M: begin array a[ 1:p(L) ];
               ........
               ........
               end
```

If the jump is taken, execution of the block labelled M is abandoned. Note that, by 5.2.4.2, L can only be outside the block (thank goodness).

5.4.5 Specifications

Incomplete specification of parameters appears to be inconsistent with the spirit of ALGOL 60, since with declarations, explicit type indications are required. Moreover, incomplete specification causes significant definition and implementation problems. The table given under 4.7.5.5 would no longer specify adequately the valid correspondence between formal and actual parameters. Hence we believe section 5.4.5 should be replaced by: 'In the heading a specification part, giving information about the kinds and types of the formal parameters must be included. In this part no formal parameter may occur more than once.'

5.4.6 Code as procedure body

In the final sentence change 'hardware representation' to 'implementation'.

Examples

As a further example of the use of ALGOL 60, the structure of the environmental block is given in detail.

EXAMPLE 3

```
begin

    comment   Simple functions;

    real procedure abs(E);
        value E;
        real E;
        abs :=
            if E ≥ 0.0 then
                E
            else
                - E;

    integer procedure iabs(E);
        value E;
        integer E;
        iabs :=
            if E ≥ 0 then
                E
            else
                - E;

    integer procedure sign(E);
        value E;
        real E;
        sign :=
            if E > 0.0 then
                1
            else if E < 0.0 then
                - 1
            else
                0;

    integer procedure entier(E);
        value E;
        real E;
```

```
    comment   entier := largest integer not greater
        than E, i.e. E - 1 < entier ≤ E;

    begin
    integer j;
    j := E;
    entier :=
        if j > E then
            j - 1
        else
            j
    end entier;


comment   Mathematical functions;

real procedure sqrt(E);
    value E;
    real E;
    if E < 0.0 then
        fault( (negative_sqrt) , E)
    else
        sqrt := E↑0.5;

real procedure sin(E);
    value E;
    real E;

    comment   sin := sine of E radians;
    <body>;

real procedure cos(E);
    value E;
    real E;

    comment   cos := cosine of E radians;
    <body>;

real procedure arctan(E);
    value E;
    real E;

    comment   arctan := principal value, in radians,
        of arctangent of E, i.e. -pi/2 ≤ arctan ≤ pi/2;
    <body>;

real procedure ln(E);
    value E;
    real E;

    comment   ln := natural logarithm of E;

    if E ≤ 0.0 then
        fault( (ln_not_positive) , E)
    else
        <statement>;

real procedure exp(E);
    value E;
```

```
    real E;

    comment   exp := exponential function of E;

    if E > ln(maxreal) then
        fault( (overflow.on.exp) , E)
    else
        <statement>;


comment   Input - output procedures;

procedure insymbol(channel, str, int);
    value channel;
    integer channel, int;
    string str;

    comment   Set int to value corresponding to the first
        position in str of current character on channel. Set
        int to zero if character not in str, unless it is
        a non-printing character, in which case set int to a
        negative integer associated with the character. Move
        channel pointer to next character;

    <body>;


procedure outsymbol(channel, str, int);
    value channel, int;
    integer channel, int;
    string str;

    comment   Pass to channel the character in str,
        corresponding to the value of int. If int is
        negative, pass the associated non-printing character,
        where the association is the same as for insymbol;

    if int = 0 or int > length(str) then
        fault( (character.not.in.string) , int)
    else
        <statement>;

integer procedure length(str);
    string str;

    comment   length := number of characters in the open
        string enclosed by the outermost string quotes;
    <body>;

procedure outstring(channel, str);
    value channel;
    integer channel;
    string str;
    begin
    integer m, n;
    n := length(str);
    for m := 1 step 1 until n do
        outsymbol(channel, str, m);
    outterminator(channel)
```

```
    end outstring;

procedure outterminator(channel);
    value channel;
    integer channel;

    comment  outputs a terminator for use after every
        string or number. To be converted into format
        control instructions in a machine dependent
        fashion. The terminator should be a space or a
        semicolon if ininteger and inreal are to be able
        to read representations resulting from outinteger
        and outreal;

   <body>;

procedure stop;

    comment  Ω is assumed to be the label of a dummy
        statement immediately preceding the end
        of the environmental block;

    goto Ω;

procedure fault(str, r);
    value r;
    string str;
    real r;

    comment  sigma is assumed to be an integer
        constant that denotes a standard output channel.
        The following calls of fault appear:
           integer divide by zero,
           undefined operation in expr,
           0.0 ↑ 0 in expn,
           undefined operation in expi,
        and in the environmental block:
           sqrt of negative argument,
           ln of negative or zero argument,
           overflow on exp function,
           illegal parameter for outsymbol,
           invalid character in ininteger(twice),
           invalid character in inreal(three times);

    begin
    outstring(sigma, `(FAULT)' );
    outstring(sigma, str);
    outreal(sigma, r);

    comment  Additional diagnostics may be output here;

    stop
    end fault;

procedure ininteger(channel, int);
    value channel;
    integer channel, int;

    comment int takes the value of an integer, as defined
```

```
        in 2.5.1, read from channel. Any number of spaces
        or other non-printing characters may precede
        the first visible character. The terminator of
        the integer may be either a space or other
        non-printing character or a semicolon (if other
        terminators are to be allowed, they may be added to the
        end of the string parameter of the call of insymbol.
        No other change is necessary);

    begin
    integer k, m;
    Boolean b, d;
    integer procedure ins;
        begin
        integer n;
        insymbol(channel, `0123456789-+.;', n);
        ins :=  if n < 0 then 13 else n
        end ins;

    for k := ins while k = 13 do
        ;
    if k < 1 or k > 13 then
        fault( `invalid character' , k);
    if k > 10 then
        begin
        d := false;
        b := k = 12;
        m := 0
        end
    else
        begin
        d := b := true;
        m := k - 1
        end;
    for k := ins while k > 0 and k < 11 do
        begin
        m := 10 * m + k - 1;
        d := true
        end k loop;
    if d impl k < 13 then
        fault( `invalid character' , k);
    int :=
        if b then
            m
        else
            - m
    end ininteger;

procedure outinteger(channel, int);
    value channel, int;
    integer channel, int;

    comment Passes to channel the characters representing
        the value of int, followed by a terminator;

    begin
    procedure digit(int);
        value int;
        integer int;
```

```
        begin
        integer j;
        j := int div 10;
        int := int - 10 * j;
        if j ne 0 then
            digit(j);
        outsymbol(channel, (0123456789), int + 1)
        end;

    if int < 0 then
        begin
        outsymbol(channel, (-), 1);
        int := - int
        end;
    digit(int);
    outterminator(channel)
    end outinteger;

procedure inreal(channel, re);
    value channel;
    integer channel;
    real re;

    comment re takes the value of a number, as
        defined in 2.5.1, read from channel. Except for
        the different definitions of a number and an
        integer the rules are exactly as for ininteger.
        Spaces or other non-printing characters may
        follow the symbol &;

    begin
    integer j, k, m;
    real r, s;
    Boolean b, d;
    integer procedure ins;
        begin
        integer n;
        insymbol(channel, (0123456789-+.&.;), n);
        ins := if n < 0 then 15 else n
        end ins;

    for k := ins while k = 15 do
        ;
    if k < 1 or k > 15 then
        fault( (invalid.character) , k);
    b := k ne 11;
    d := true;
    m := 1;
    j :=
        if k < 11 then
            2
        else
            iabs(k + k - 23);
    r :=
        if k < 11 then
            k - 1
        else
            0.0;
    if k ne 14 then
```

```
        begin
        for k := ins while k < 14 do
            begin
            if k < 1 or k = 11 or k = 12
                or k = 13 and j > 2 then
                fault( (invalid.character) , k);
            if d then
                begin
                if k = 13 then
                    j := 3
                else
                    begin
                    if j < 3 then
                        r := 10.0 * r + k - 1
                    else
                        begin
                        s := 10.0↑( - m);
                        m := m + 1;
                        r := r + s * (k - 1);
                        d := r ne r + s
                        end;
                    if j = 1 or j = 3 then
                        j := j + 1
                    end
                end
            end k loop;
        if j = 1 and k ne 14 or j = 3 then
            fault( (invalid.character) , k)
        end;
    if k = 14 then
        begin
        ininteger(channel, m);
        r := (if j = 1 or j = 5 then 1.0 else r)
             * 10.0 ↑m
        end;
    re :=
        if b then
            r
        else
            - r
    end inreal;

procedure outreal(channel, re);
    value channel, re;
    integer channel;
    real re;

    comment Passes to channel the characters representing
        the value of re, followed by a terminator;

    begin
    integer n;
    n := entier(1.0 - ln(epsilon) / ln(10.0));
    if re < 0.0 then
        begin
        outsymbol(channel, (-), 1);
        re := - re
        end;
    if re < minreal then
```

```
        begin
        outstring(channel, (0.0) );
        end
    else
        begin
        integer j, k, m, p;
        Boolean float, nines;
        m := 0;
        nines := false;
        for m := m + 1 while re ≥ 10.0 do
            re := re / 10.0;
        for m := m - 1 while re < 1.0 do
            re := 10.0 * re;
        if re ≥ 10.0 then
            begin
            re := 1.0;
            m := m + 1
            end;
        if m ≥ n or m < - 2 then
            begin
            float := true;
            p := 1
            end
        else
            begin
            float := false;
            p :=
                if m = n - 1 or m < 0 then
                    0
                else
                    m + 1;
            if m < 0 then
                begin
                outsymbol(channel, (0), 1);
                outsymbol(channel, (.), 1);
                if m = - 2 then
                    outsymbol(channel, (0), 1)
                end
            end;
        for j := 1 step 1 until n do
            begin
            if nines then
                k := 9
            else
                begin
                k := entier(re);
                if k > 9 then
                    begin
                    k := 9;
                    nines := true
                    end
                else
                    re := 10.0 * (re - k)
                end;
            outsymbol(channel, (0123456789), k + 1);
            if j = p then
                outsymbol(channel, (.), 1)
            end j loop;
        if float then
```

```
            begin
            outsymbol(channel, (&), 1);
            outinteger(channel, m)
            end
        else
            outterminator(channel)
        end
    end outreal;


comment   Environmental enquiries;

real procedure maxreal;
    maxreal := <number>;

real procedure minreal;
    minreal := <number>;

integer procedure maxint;
    maxint := <integer>;


comment   maxreal, minreal, and maxint are, respectively
    the maximum allowable positive real number, the
    minimum allowable positive real number, and the
    maximum allowable positive integer, such that any
    valid expression of the form
        <primary><arithmetic operator><primary>
    will be correctly evaluated, provided that each of the
    primaries concerned, and the mathematically correct
    result lies within the open interval (-maxreal,-minreal)
    or (minreal,maxreal) or is zero if of real type, or within
    the open interval (-maxint,maxint) if of integer
    type.
    If the result is of real type, the words 'correctly
    evaluated' must be understood in the sense of
    numerical analysis (see Revised Report 3.3.6);

real procedure epsilon;

    comment   The smallest positive real number such that the
        computational result of 1.0+epsilon is greater than 1.0
        and the computational result of 1.0-epsilon is less than
        1.0;

    epsilon := <number>;

comment In any particular implementation, further
    standard functions and procedures may be added here,
    but no additional ones may be regarded as part of the
    reference language;

<fictitious declaration of own variables>;
<initialisation of own variables>;

<program>;

 Ω :
end
```

Notes on the standard procedures and functions

The above coding is only to be taken as definitive in terms of its effect on correct programs, ignoring those questions which are the domain of numerical analysis. For instance, a call of the procedure 'fault' indicates that the program is in error, and hence after detection of the error, different action may be taken than that indicated by the above coding. Actual implementations may produce better diagnostics than are possible to express conveniently in ALGOL 60.

The procedures sin, cos, arctan, ln, and exp have some coding omitted because their definition is clear and this report is not concerned with the methods used in the evaluation of these functions. The bodies of the procedures insymbol, outsymbol, length, outterminator, maxreal, minreal, maxint and epsilon are omitted because of their obvious machine dependence. The procedures insymbol and outsymbol are used on the assumption that the relevant 'ALGOL basic symbols' are single characters. Appropriate changes must be made if this is not the case, although the only likely exception is the use of & in 'inreal' and 'outreal'.

Naturally, implementations should gain significantly in performance over the coding given above. In particular, the simple functions may be performed by open code, the variable n in outreal can be assigned the appropriate constant value, the procedure identifiers maxreal etc can be replaced by a constant value and the recursive nature of the procedure digit can be avoided. Also, the numeric properties of the procedures inreal and outreal can be enhanced by the use of double length working, although these procedures have been tested and found to be adequate (within the constraints of single precision).

Index

The following corrections should be made to the index of the Revised Report:-

. delete entry to conform with amendments.

<arithmetic expression> delete 'synt 3.3.1' as this appears under def.

<array declarer> add entry containing 'def 5.2.1'

<local or own type> delete entry.

<procedure identifier> insert 4.2.1 under synt.

<simple arithmetic expression> insert 'synt 3.4.1'.

space delete 'def 2.3'

<type> add 'synt 5.2.1'

<unsigned integer> delete '3.5.1.'

<variable> delete '4.6.1,'

<variable identifier> insert 'synt 4.6.1'

References

The documents used to construct this commentary are too numerous to list, but the principle references are:

[1] Naur, P (Editor) Revised Report on the Algorithmic Language ALGOL 60,
Comm ACM, Vol 6 (1963), p1
Comp J, Vol 5 (1963), p349
Num Math, Vol 4 (1963), p420

[2] Report on Subset ALGOL 60 (IFIP),
Num Math, Vol 6 (1964), p454
Comm ACM, Vol 7 (1964), p626

[3] ECMA Subset of ALGOL 60,
Comm ACM, Vol 6 (1963), p595
European Computer Manufacturers Association (1965) ECMA Standard for a Subset of ALGOL 60.

[4] ISO/R 1538, Programming Language ALGOL (1972)

[5] Report on Input-Output Procedures for ALGOL 60 (IFIP),
Num Math, Vol 6 (1964), p459
Comm ACM, Vol 7 (1964), p628

[6] Knuth, D.E. et al, A Proposal for Input-Output Conventions in ALGOL 60,
Comm ACM, Vol 7 (1964), p273

[7] Knuth, D.E. The Remaining Trouble Spots in ALGOL 60
Comm ACM, Vol 10 (1967), p611

[8] Suggestions on the ALGOL 60 (Rome) Issues,
Comm ACM, Vol 6 (1963), p20

[9] A booklet on ALGOL 60,
Joint IFIP/NPL Publication, to be prepared.