AB20.2 CORRESPONDENCE

AB20.2.1 Comments on the proposals of H. Rutishauser, AB19.3.10

Norwegian Computing Center,
Forskningsveien 1 b,
Oslo 3, Norway.

Dear Editor,

I have just read AB19, and in particular, the paper by Mr Rutishauser, AB19.3.10, caught my interest.

I have a feeling that if the word long is used to denote double precision, we leave no doors open for expansion if the manufacturers of computers decide that they will include triple precision or even multiple precision arithmetic in their computers' hardware.

I would suggest (as probably a lot of others have done before because a restricted version of this method is used in UNIVAC 1107 ALGOL) that the syntax for <local or own type> is changed to

<local or own type> ::= <type designator> | own<type designator>

and that

<type designator> ::= <type> | <type><unsigned integer>

is added to the syntax. The clause <unsigned integer> denotes the precision in number of words, and could for instance be single precision if the unsigned integer was zero or one.

To declare a triple precision real number, the declaration would be for instance:

real 3 a, b, c;

The lexico concept introduced by Mr Rutishauser will probably be unnecessarily complicated both for implementors and programmers. It would be an operation on identifiers rather than values. If my interpretation of 4.7.3.2 in the revised ALGOL report is correct, then the following complication will arise:

begin real a, b, c, d, e, z;
    procedure q(a, b); real a, b;
    begin list 1 := a lexico b; end;
    q(a, z);
end

When the procedure is entered the list 1 would change its
appearance to \texttt{list 1 := a lexico z}
because of 4,7,3,2 and even further complications will arise if the
procedure body was changed to
\begin{verbatim}
begin real z; list 1 := a lexico b; end;
\end{verbatim}
I also disagree with Mr Rutishauser on a number of other proposals,
but since this is supposed to be a letter to the editor, I hope that
somebody else will comment on these.

Björn Myhrhaug

\section*{Two short paragraphs}

28 Howard Road,
Leicester, England.
March 17th 1965.

Below are two short paragraphs for AB20.

(1) \textbf{On C.A.R. Hoare's Case Expressions}

I too like this idea put forward by C.A.R. Hoare (AB16,3,7).
However the syntax of case statements leads to ambiguities,
e.g. \texttt{case n of begin if BE then S1 else S2 else S3 end}
Is this equivalent to
\begin{verbatim}
case n of begin
    begin if BE then S1 else S2 end
    else S3
end
\end{verbatim}
or to
\begin{verbatim}
case n of begin
    begin if BE then S1 end
    else S2
    else S3
end
?\end{verbatim}

(2) \textbf{WG21 and ALGOL X}

I understand that WG21 is hoping to define ALGOL X at its
next meeting in Princeton in May.
Up to AB19 there have only been a few comprehensive suggestions. After discussion, compromise and agreement has been reached I feel that ALGOL X will contain ambiguities, inconsistencies, and side-effects (undesirable even if they do allow some fascinatingly tricky programs), if it is defined this May.

For this reason I will be happier if WG 2.1 limits itself to a set of suggestions for ALGOL X and decisions only on such topics as:

- AB18.3.1 H.C. Thacher Jr.'s "On mixed types in relations"
- AB19.3.3 P.Z. Ingerman's "Blemishes"
- AB19.3.7 D.E. Knuth's "Remaining Trouble Spots".

R. S. Scowen

P.S. I like NPL (the place not the language).

---

Concerning "Teaching ALGOL60"

Research Institute of Mathematical Machines,
Loretánské n. 3,
Prague 1,
Czechoslovakia.

March 16th 1965

I beg to submit the following correction to an item in no. 19 of the ALGOL Bulletin.

In AB19.2.1, D. Knuth "Teaching ALGOL60", the objection to item 1 is irrelevant: no extraneous meaning should be imputed to the precise wording of the ALGOL Report 5.2.4.2. Indeed, the declaration array M[1: I] would be perfectly legitimate if the assignment I := 0 were to precede, and not follow, the block containing this array declaration.

Otomar Hâjek
AB20.2.4 Seven times faster

National Institute for Research in Nuclear Science,
Atlas Computer Laboratory,
Chilton, Didcot, Berks., UK.

April 7th 1965

Readers may be interested in run-times of the Knuth Test [AB17.2.4] on the I.C.T. Atlas ALGOL Compiler.

The test made by Price and Hodg~ [AB19.2.3.1] for k = O (1) 11 was repeated on the Atlas and a run time of 1.75 seconds was obtained which compares with the 12 seconds using the KDF9 Kidsgrove Compiler.

F.R.A. Hopgood

[Other run-times reported have included 20 minutes and 80 (sic) minutes. The reporters prefer to remain anonymous, as also do their machines. -Ed.]

AB20.2.5 ECMA and the standardisation of ALGOL

European Computer Manufacturers Association,
rue d'Italie 11,
Geneva.

April 7th 1965

Please find enclosed [reproduced on the following pages - Ed.] a copy of document ECMA/TC5/65/4, "ECMALGOL and the Implementation of a standard for ALGOL 60". This document has been prepared by the Technical Working Committee TC5 of ECMA and gives the reasons for which it has been felt necessary to promulgate ECMALGOL as an ECMA Standard. I would be very happy if you would publish this paper in AB20.

D. Hekimi,
Secretary General, ECMA.
"ECMALGOL" and the Implementation of a Standard for ALGOL60

1. Introduction

In April 1964 the General Assembly of ECMA adopted as a standard the so-called "ECMALGOL", containing the definition of a subset of ALGOL60, produced by the ECMA Technical Committee on ALGOL, TC5.

In May 1964, the Council of IFIP approved for publication the so-called "SUBSET ALGOL60 (IFIP)", a subset of ALGOL60 defined by the IFIP Working Group on ALGOL, WG 2.1.

The two subsets are not identical; however, the latter is contained entirely within the former.

At the end of May 1964 the ISO Programming Languages Subcommittee (ISO/TC97/SC5) arrived at a number of provisional decisions concerning the drafting of a first proposal for the standardisation of ALGOL60. Among these was that the draft would contain the text of the Revised ALGOL60 Report together with that of a "unique subset of ALGOL60" based on the IFIP subset. In other words, the language ALGOL60 would be standardised at two levels, and two levels only, namely that of the language defined by the complete ALGOL60 Report, and that of the language defined by the IFIP subset report.

A minority of the ISO subcommittee favoured the recognition of a number of "nested subsets" or "levels of implementation". The members of ECMA TC5 support this view.

It is the purpose of the present document to answer a number of questions arising out of the situation briefly described above. Among these are:

(1) Why was ECMALGOL defined?
(2) Why is ECMALGOL not identical with ALGOL60?
(3) Why is ECMALGOL not identical with SUBSET ALGOL60 (IFIP)?
(4) What does ECMA imply by adopting ECMALGOL as a standard?
(5) Should an ISO draft obtain recognition as a standard, will there be any reason for ECMALGOL to continue to exist as an ECMA standard?
(6) What is the view of ECMA as to the nature and force of a programming language standard, and in particular, what should be the effect on the members of ECMA of the existence of an ECMA standard, and of an ISO standard?

2. Background

Before any attempt is made to answer these questions it is as well to recall the history of ALGOL60 and its subsets.

Soon after the publication of ALGOL60 questions and criticisms began to appear in the pages of the ALGOL Bulletin and elsewhere, but no united answer to any of them was given until April 1962. In the
meanwhile a number of groups began to make, and to use, compilers for ALGOL. Outstanding among these was the ALCOR Group, which included many German academic institutions and computer manufacturers. The aims of this group were extremely practical; the compilers for the different machines were made from the same plans and compatibility to the level of a common hardware representation was realised. The group accepted a number of restrictions to the language; some of these were consequent on the compiling method used, some were forced by machine limitations, and some arose out of the group's interpretations of the language.

The first ALGOL60 compiler actually completed came from outside the ALCOR Group; its methods were such that an almost unrestricted implementation was possible - furthermore, the criterion for interpreting the Report was not "What did the authors intend to write here?", but "What are the consequences of the literal meaning of the words they have written?".

With the exception of the German companies of the ALCOR Group, the attitude of the TC5 members was somewhere between these two views. Their compilers were, typically, compromises between the implications of the language as defined, the requirements of the users, and the budgets within which they were worked out. The primary object of TC5, which first met in early 1962, was to establish the extent to which ALGOL was indeed being implemented, or could be implemented, and so to define any restrictions that were necessary before its adoption as a common programming language.

The IFIP ALGOL Group, which first met in August 1962, was, and is, quite different from TC5 in object. It is the inheritor of the work and responsibilities of the authors of ALGOL60, and as such its scope is the "development, specification, and refinement of the ALGOL language". It is not charged with maintenance, or with the establishment of conventions as to the use, or methods as to the implementation of ALGOL60. It has, however, been asked by ISO to produce a subset of ALGOL60 avoiding the so-called "ambiguities" and in reply has produced the IFIP Subset of ALGOL60. The present work of the IFIP ALGOL Group is concerned, not with ALGOL60, but with more advanced languages made possible by the experience of ALGOL60.

3. Motivation of the Subsets

The motives leading to the publication of the IFIP and ECMA subsets have already been hinted at.

The most concise statement of the reason for the definition of the ECMA subset was contained in the original terms of reference of TC5: "... the extent to which" ALGOL had been implemented or could be implemented. Behind these words is the assumption that the members of ECMA would have liked to have been able to accept the whole of ALGOL60 as a practical standard, but that there were difficulties of one kind and another. Thus the ECMA subset is the largest subset of ALGOL60 which, in the opinion of the members collectively, can be implemented.
for practical use. It follows that the reasons for the several restrictions are practical, rather than aesthetical or critical. In Appendix I an explanation is given of each one by itself. It should be noted that restrictions 1 and 8 avoid difficulties in interpreting the language; 5, 6 are there chiefly to avoid hardware difficulties; 2, 3, 4, 7 are restrictions which permit faster object programs in the most frequent applications of the features concerned.

The IFIP subset was to a large extent compromise between the ALCOR version of ALGOL and the American SMALGOL. From the IFIP subset report itself we learn that its objectives include:

1. the avoidance of certain unwanted generality of ALGOL60;
2. the avoidance of controversial parts of the language;
3. the unification of a number of different subsets, already in use.

It is further stated in the "General Introduction" to the ISO first draft proposal that this subset was intended to be "implemented more easily" than ALGOL60 and to be moreover "the minimum feasible subset".

With regard to the last point it is certainly true that some restrictions were made not because any difficulty of implementation or any inefficiency was involved but because it was felt that programmers had no great need of the feature concerned. Such restrictions are (see Appendix II) 9, 12, 14. Controversial facilities are suppressed by restrictions 9, 12, and 15. Restriction 10 seems somewhat arbitrary, while 16 is a formality of no practical significance to ALGOL60 users - it merely prevents the writing of certain procedure declarations which are either trivially redundant or unable to be called unambiguously.

Restrictions 1 to 8 are common to both subsets. They were originally drafted by TC5 and then adopted by WG 2. The other restrictions were all rejected by TC5 when it was suggested that the ECMA subset might be restricted to coincide with the IFIP intentions. The foregoing explanations should at least indicate the reasons for leaving the ECMA subset as it was.

4. Criticisms of ECMALGOL

Recent criticisms of ECMALGOL can be summarised under the following four headings:

1. It is too restricted.
2. It is not restricted enough.
3. It is not a subset of ALGOL60.
4. It is a subset of ALGOL60.

Specific examples are:

1. Objection: ECMALGOL would be more useful if recursion were allowed.
   Answer: At the time the subset was defined, recursion was not, and could not, be implemented in all ECMA compilers. Since the subset has been approved as an ECMA standard a number of members have begun working on compilers without recursion.
   Remark: It is important to remember that the standard allows compilers to provide more than the subset. Thus there are ECMA
compilers which allow recursion; however, programs which exploit recursivity cannot be used with these other compilers without serious modification. The same applies to other restrictions. See also further remarks below.

(2) Objection: Since the official Working Group on ALGOL has defined a subset of ALGOL60 there is no need for an ECMA subset at all; therefore ECMALGOL should be made identical with the IFIP subset.
Answer: The objectives of the two subsets have already been mentioned and contrasted. It would be a retrograde step to adopt a standard lower than the apparent practical level of implementation and application. The reasons for not accepting the additional restrictions of the IFIP subset are given in Appendix II, with emphasis on the problem of rewriting an (unrestricted) ALGOL program to conform to the several restrictions.

(3) Objection: The "relaxation of type requirements" (7) and the ignoring of any symbols in an identifier after the first six (6) are not true restrictions, but rather changes to the language. With these "restrictions", it is possible to write a valid subset program whose meaning is different when regarded as an ALGOL60 program.
Answer: This objection applies equally to the ECMA and IFIP subsets. The assumption behind restriction 7 was that the integers were a subset of the reals. If this is not granted, then the idea of "relaxation of type requirements" becomes meaningless. The objection to restriction 6 is based on a misinterpretation of the word "undefined". This word must be taken at its face value; it does not mean "nonexistent" or "indeterminate".

(4) Objection: By making an ECMA programming language standard a subset of ALGOL60, ECMA has perpetuated the disadvantages of ALGOL60 as a programming language.
Answer: This objection has been put both as regards details and as regards the language as a whole. As to the first, concerning specifically strings and variable list parameters, it should be pointed out that the intention was not to propose extensions to the ALGOL language which might not have been generally acceptable, but to show how much of the existing language might reasonably be used as a practical standard. As to the second, the existence of an ECMA standard for ALGOL is in no way intended to preclude the adoption of standards for any other programming language.
Appendix I - Restrictions common to ECMA and IFIP

1. **own excluded**

   The concept of *own* is not well defined in the Report; the Rome conference was unable to improve the definition. Except on the lowest level different but equally plausible interpretations are possible. Each of these leads to a degree of clumsiness, both in the compiler and in the object program, which in many situations is not felt to be justified by the additional facility in the language. If recursion (see 2 below) is excluded, it is neither difficult nor inefficient to have *own* simple variables and fixed-bound arrays. If recursion is admitted, however, it is often better not to have *own* at all; those who support the idea of the levels are unanimous in the view that recursion should be admitted before *own*.

   Thus "ECMALGOL + recursion"

   and

   "ECMALGOL + recursion + (restricted) own"

   might be recognised, but not

   "ECMALGOL + (restricted) own".

2. **Recursion excluded**

   The decision as to whether a compiler will provide for recursion or not has to be made very early in its design; the question is fundamental to the structure of both the compiler and the object program. It is very difficult to modify a "non-recursive" compiler to make it deal with recursion. A number of non-recursive compilers are in use within ECMA and others are being made.

   Although a recursive compiler can be no more difficult to make and no bigger than a non-recursive compiler for the same machine, such recursive compilers will produce, for many machines, unacceptably slow object programs. Recursive compilers to produce sufficiently fast object programs may require considerably more work, some concerned with the language itself and some with the characteristics of the computer; this additional work is often felt to be unjustified, on economic or other practical grounds.

   Recursive compilers of both kinds exist within ECMA, sometimes alongside non-recursive compilers for the same machines at the same installations.

   It is important, therefore, for the subset to exclude recursion. However, there is a wide range of application for which recursion is desirable, and there are ECMA compilers which provide this facility.

3. **Integer labels excluded**

   This is a very common restriction. It causes no inconvenience to the programmer. There appear to be no cases where the Report allows the essential use of the fact that a particular label is an integer. If integer labels are allowed, object programs must allow for the possibility that an unsigned integer occurring as an actual parameter may be either an (arithmetic) integer or a label.

4. **Insistence on specification**

   Formal parameters called by value must always have specifications. In full ALGOL60 formal parameters called by name need not; consequently
situations occur in which the compiler must generate alternative pieces of object program, or else become involved in analysis (more or less elaborate and successful according to the nature of other restrictions) as to what actual parameters are likely to enter into the procedure calls.

If a specification is insisted on for each formal parameter, then this specification can be used to select only one of the possible alternatives for the object program version of the procedure declaration. For each call of the procedure, the actual parameters must now be such that the transformed body is a "correct ALGOL statement", and moreover their kinds and types must be "compatible" (see section 4.7.5) with those of the formal parameters. In the absence of a definition of compatibility (see the last paragraph of the Introduction to ECMA/ALGOL) the programmer must assume that parameters of identical kind and type are compatible. He cannot assume in general that real and integer are compatible, as the following example will show:

```plaintext
begin real x, y, z; integer l, m, n;

procedure IntMult(i, j, k); integer i, j, k; i := j \times k;
m := 2; n := 1; y := 2.49; z := 1.49;
IntMult(l, m, n); comment certainly l := 2;
IntMult(x, y, z); comment In view of 4.7.5 and 4.7.5.5 this
call should be regarded as undefined.
It is in any case not clear whether
it would assign 2 or 4 to x;

end
```

5. **Alphabet one case only**
   All the relevant equipment caters for at least 26 letters; only a few devices can manage 52 satisfactorily.

6. **Only six symbols of identifier distinguished**
   This restriction, which is not serious for programmers warned of it in advance, allows many compilers to treat each identifier in a single machine word or other fixed-length field. In practice some limitation on length of identifier must exist and the limit of six symbols has been found by experience to be adequate.

7. **Restriction of type requirements**
   In ALGOL60, some expressions (e.g. involving if clauses or exponentiation), and consequently some operations, are indeterminate as to type until they are evaluated. Therefore the compiler may have to be made to generate alternative sets of instructions in the object program. With the restriction all types can be determined from the source program alone, and so the object program can be both shorter and faster. The effect on the programmer depends on the way in which the arithmetic types have been implemented. If real and integer are distinct (as is the case when they are fixed- and floating-point numbers, in many machines) and one wishes to work sometimes with reals and sometimes with integers, it is necessary to program two routines where unrestricted
Go to undefined switch designator undefined

It is almost generally agreed that the Report is perverse in prescribing that a go to statement leading to an undefined switch designator should be equivalent to a dummy statement. In good programming practice such a statement would be regarded as an error, or the situation would be deliberately avoided. For example, if it were known that $i$ could take values outside the range 1 to $n$ instead of simply go to $S[i]$ one would write:

$$\text{if } 0 < i \land i < n \text{ then go to } S[i];$$

It is in fact impossible to have the "dummy statement" effect without also having a restriction against side-effects. Consider the example:

```plaintext
begin integer procedure i; begin output(3); i := 0 end;

procedure output(x); real x;

  comment prints value of x; <code>

switch S := L;

L: go to S[i]

end
```

Is there any implementation that can avoid printing "3"? The point is that before it is determined that the subscript is outside the bounds, an irreversible side effect may have been involved in the evaluation. Even though IFIP excludes side effects, it makes the same improvement to the language, but in a more pointed manner, by omitting the original statement altogether.
Appendix II - IFIP restrictions not in the ECMA subset

9. Side effects excluded
   This has always been a very controversial issue. It must be pointed out, however, that much of the controversy has been fruitless. Since the order of evaluation of the primaries of an ALGOL expression is undefined, many of the examples used in earlier discussions are ambiguous - in particular those in which the evaluation of a function designator alters the value of a variable occurring elsewhere in the same expression. Those side effects which are well-defined in ALGOL60 (such as using a non-local variable which does not occur elsewhere in the expression for counting the calls on the function or for recording the value of some intermediate result) do not introduce new difficulties of implementation, nor do they cause conflicts between the notation of expressions and that of "normal mathematical usage".

10. Exponentiation restricted
   The restriction is rather more than is necessary to ensure that the type of any factor can be determined by the compiler. For the programmer the rules are rather complicated, as is shown by the following lists:
   (i and j are assumed declared integer)
   Invalid:  i ↑ (2), i ↑ (2), i ↑ (-2), 2 ↑ (-27), 2 ↑ j, i ↑ j
   Valid:    i ↑ 2, i ↑ 2.0, i ↑ (-2.0), 2 ↑ (-27.0), 2.0 ↑ j, i ↑ (j + 0.0)
             i ↑ (+2.0), (i/1) ↑ (2), 2 ↑ 27, 2.0 ↑ (-27), 2 ↑ (j + 0.0), (i + 0.0) ↑ j
   Of those invalid in the subset, the first two are defined as integer by ALGOL60, while the next two are defined real (except of course for i = 0). Only the last two are of variable type, depending on the value of j. Here, restriction 6 (common to ECMA and IFIP) allows the type to be taken as real for all values of j. Thus the effect of this IFIP restriction is to exclude a number of cases whose types are no more difficult to establish than are those of the admitted cases, and moreover to introduce new difficulties of checking for the compiler writer and the programmer. Overall, the situation is made worse, not better.

11. No integer division
   The ALGOL60 statement i := j * k;
   has to be replaced by
   begin real x; x := j/k;
             i := sign(x) × entier(abs(x)) end;
   which will surely require much more compiling and execution time.

12. Controlled variable to be simple variable
   With this restriction the compiler can hold a single register for the controlled variable during the execution of a for statement (provided, of course, that such a register is available); at object time the address of the controlled variable does not change. Shorter and
faster object programs are thus sometimes possible for the allowed case, but perhaps not so short and fast as to justify the restriction. For significant gains in speed without an elaborate compiling mechanism, more drastic restrictions would be needed, which would, however, still allow the very common simple cases typified by "for i := 1 step 1 until n do S". Even then, for full advantage, restrictions would have to be put on the controlled statement S concerning the occurrence of the controlled variable i and the behaviour of procedure parameters.

An ALGOL60 program making use of the forbidden feature is best rewritten in the subset by means of the expanded forms given in section 4.6 of the Revised ALGOL60 Report.

13. **Conditional designational expressions excluded**

Designational expressions may occur (i) in go to statements, (ii) in switch lists, (iii) as actual parameters.

The restriction can be overcome in the first two cases as follows:

(i) Replace:  
go to if B then L1 else L2
   
   by:
   
   if B then go to L1 else go to L2

(ii) The switch list can be extended to include all the possible labels and the evaluations of the Booleans transferred to the evaluation of the subscript of the switch designator.

E.g. replace:  
switch S := S1, S2, Q[m], if v > -5 then S3 else S4;

switch Q := p1, w; ...;
go to S[i];

by:

switch S := S1, S2, p1, w, S3, S4;

integer procedure j;

\[
\text{begin if i = 1 then } j := 1; \\
\text{if i = 2 then } j := 2; \\
\text{if i = 3 then begin if } m = 2 \text{ then } j := 4 \\
\text{else if } m = 1 \text{ then } j := 3 \text{ end; } \\
\text{if i = 4 then begin if v > -5 then } j := 5 \\
\text{else } j := 6 \text{ end }
\]

end; ...;
go to S[j];

(iii) It is difficult to suggest rules for the rewriting of actual parameters - consider for example

P(b, if b then L1 else L2)

where both parameters are called by name. The safest approach is to rewrite the whole procedure. [Incidentally, IFIP also forbids this example by restriction 15.]

The effect of the restriction on object program speeds for the allowed cases depends very much on the compiling method used. For the more general cases which have to be rewritten at greater length any improvement in
speed may well be nullified; indeed in some cases longer times are possible.

14. **Switch elements labels only**

This means that switch designators are excluded from switch lists. To overcome this, a program would have to be rewritten by replacing any switch designator occurring in a switch list by the (explicit) switch list on which it operates, and the "label selecting algorithm" would have to be rewritten. Cf. the examples under 13 above.

15. **Call by name restricted**

This is perhaps the most important of the restrictions in the list. Among its effects on programming are:

(i) A subscripted variable cannot be written as a "result" parameter.

(ii) Jensen's device is made impossible.

Examples:

(i) for \(i := m\) step 1 until \(n\) do read(\(A[i]\));

is not allowed, but has to be expressed:

for \(i := m\) step 1 until \(n\) do begin read(\(x\)); \(A[i] := x\) end;

(ii) The ALGOL Report example "Innerproduct" with the call

\[\text{Innerproduct}(A[t, P, u], B[P], 10, P, Y)\]

would have to be rewritten with the name parameters as function procedures:

```
begin procedure Innerproduct (a, b, k, p, y); value k;
  integer k, p; real procedure a, b; real y;
  begin ... end Innerproduct;
  real procedure f11; f11 := A[t, P, u];
  real procedure f12; f12 := B[P]; ...;
  Innerproduct(f11, f12, 10, P, Y); ... end
```

Perhaps a more convincing example is

\[\text{Innerproduct}(A[i, j], B[j, k], n, j, C[i, k])\]

Here the "result" parameter \(C[i, k]\) is very awkward to handle. The "obvious" transformation, specifying "procedure \(y\)" and replacing "\(y := s\) end" by "\(y\) end", with "procedure \(f23; C[i, k] := s\)" is not correct, because of the problem of the scope of \(s\).

The trouble in all these examples is in the subscripted variables; an array has a name, but a subscripted variable has not.

While the value call can be described in terms of more primitive ALGOL concepts, and is indeed so defined in the Report, the name call cannot. The central idea of the explanation given in the Report on call by name is that of the transformation of the program text itself; this
concept does not exist outside the actual-formal substitution of the call by name. It follows that restrictions to call by name remove unique features from the language. Thus, in general, programs which have made use of the full call by name facility cannot be rewritten in a subset which restricts the call by name.

16. Double parameters forbidden

This is a "dummy restriction". It only prevents the writing of ambiguous procedure declarations, undefined procedure statements, and redundant parameters.
Appendix III - References to relevant documents

Defining documents
ECMA: Standard for a subset of ALGOL 60 (1964)

Discussion of language
ALGOL Bulletin 10-15, Regnecentralen, Copenhagen, for ALGOL 60.
ALGOL Bulletin 16- , Mathematisch Centrum, Amsterdam,
for Revised ALGOL 60.

Implementation
DIJKSTRA, E. W. ALGOL 60 Translation (1961)

Miscellaneous
ECMA/TC2/64/11: Requirements for a programming language.
ISO first draft proposal on ALGOL 60:

March 1965.