Naur 3
Korrektur kopi

REGNECENTRALEN
Gl. Carlsbergvej 2,
Valby, Copenhagen,
Denmark.
2. Jan. 1960.

Material related to the coming

INTERNATIONAL ALGOL CONFERENCE IN PARIS,
January 11. - 16. 1960.

Enclosed I send you some material related to the subject of the coming ALGOL conference. It is divided into sections as follows:

1) Notes and explanations concerning the proposals of the European participants at the conference as expressed in the MAINZ REPORT from the meeting of Dec. 14. - 16. 1959.
2) Suggestions in the form of drafts for certain sections of the final report (to be prepared during the coming meeting).

It should be noted (A) that the opinions expressed and formulations chosen are a mixture of my own and those of the other European participants, (B) that where a draft for the final report on controversial questions is presented it is not with the aim of stopping a fundamental discussion before it starts, but only in order to help formulating the final report, and (C) syntax has in a few places been expressed in the notation used by Backus (Paris report June 1959), using the symbol ::= for Backus' := and | for Backus' or. Elements not defined in my text are identical with the same elements as defined by Backus.

Peter Naur
Peter Naur.

To:

John W. Backus,	New York
F.L. Bauer,	Mainz
McCarthy,	Cambridge
Julien Green,	New York
C. Katz,	Philadelphia
Alan J. Perlis	Pittsburgh
H. Rutishauser,	Zurich
K. Samelson	Mainz
W. Turanski,	Philadelphia
B. Vauquois,	Grenoble
J. Wegstein,	Washington
R.A.v.Wijngaarden,	Am sterdam
M. Woodger,	Teddington
P. Naur,	Copenhagen

SUGGESTIONS AND CORRECTIONS TO MAINZ REPORT.

1. Retain the delimiters for exponentiation (i.e. delete Mainz report section 5, first line).

2. Write declarations governing a compound statement after begin. This in order to make procedure declarations, including their own compound, appear in their most natural place. In addition the change relative to the Zurich report becomes less pronounced.

3. Mainz report section 1. Declarations governing a Statement, add: >>In the lower-upper-bound lists of array declarations only integer valued expressions in variables, which are global to the statement will occur. These expressions will be evaluated, and the array declarations thus completed, upon entrance into the statement.
The formal variables of a procedure declaration must in this connection be considered global to the compound of the declaration<<.

4. Mainz report page 3, add one line of text between lines 8 and 9 as printed, to read:
>> . . . . . .output parameters. These are to be constructed in the following manner. The group D1 must give . . . . . <<.

5. Mainz report page 3, in list of possible types of information, add:
type array (I, I, . . . I[d:d], ...)
type function (. . . . .)
input (I(d,d,...,d), I(d,d,...,d), ...)
output (I(d,d,...,d), I(d,d,...,d), ...).

6.1. Treat the for-statement as a clause (cfr. Backus).
6.2. Permit declarators for type.
6.3. Treat expression list like the list of actual parameters in a procedure statement.

7. The proposed constant declaration is not made obsolete by the >>functions without parameters<<, since this does not cater for constant arrays.

8. Confine the appearance of arbitrary strings of symbols to format declarations (i.e. disallow them as parameters for procedures).

THE SYNTACTICAL STRUCTURE PROPOSED BY THE EUROPEAN MEMBERS.

Some of the reasoning leading to the proposal of the European members is the following:

1. If array declarations with subscript ranges depending on expressions in running variables are to be permitted the concept >>dynamic declaration<< must necessarily be cleared up syntactically.

2. It is realized that in principle a range-limiting by write-up or time-succession may be chosen at will. If array-declarations were governed by the time-succession additional rules for the exact semantic meaning of changing the dimension or subscript range of a given array become necessary. In a range-limiting strictly by write-up, with the additional rule that arrays may be either defined or cancelled, but never jutt changed, this difficulty is avoided.

3. This ties in with the suggestion that the meaning of identifiers in general should be limited by write-up (local-concept). This idea is already present in the procedures (internal variables). One is thus led to the idea that the meaning of all declarations should be limited by write-up.

4. Unless an arbitrarily large set of new brackets be introduced, the range limiting must be attached to the existing brackets begin end.

5. In order to couple the different levels, defined by the begin-end structure (begin increases the level, end decreases it) it is necessary to allow that certain entities are retained across the begin-end boundaries. For simplicity it is suggested that such quantities which are global to a certain level (i.e. common to this level and the next lower level), need not be declared at all, in agreement with the Zurich report.

6. It seems highly desirable to limit the admissible jumps from one level to another, since an arbitrary jump will in general make the cancellation of the quantities from several levels and the simultaneous definition of the quantities from several other levels necessary. The restriction >>all labels are local<< implies that dynamically an incr ase of level must always happen by a passage through a begin. A decrease in level, on the other hand, may pass through several levels at one time.

THE EUROPEAN PROPOSAL FOR PROCEDURES.

The European proposal means a restriction of the Zurich language in the following respects:

1. Only one procedure may be defined in any one procedure declaration.

2. Partly empty parameter positions in procedure statements are disallowed.

On the other hand, the following advantages have been achieved:

3. The necessity for empty bracket skeletons has been removed.

4. A procedure heading will give complete information (including types of parameters of formal input procedures) concerning the formal parameters.

5. Functions and procedures without parameters become possible.

6. All functions are defined through function declarations (of which there are two types: short-hand and procedure-like).

7. Procedure declarations become very similar in structure to other compound statements.

Some of the considerations leading to this result are the following:

A. If 4. is to be achieved either a) an even more complicated symbolism in the procedure heading than envisaged in the Zurich report becomes necessary, or b) an entirely different notation must be invented. Since the empty brackets were felt to be already too unwieldy the possibility b) was explored, with the result described in our report.

B. Since in this way the necessity for bracket skeletons is avoided in procedure headings, the possibility of avoiding them in procedure statements as well may be explored. The difficulty here is the possibility of >>partly-open<< bracket structures envisaged in the Zurich report, viz. partly functions and procedures and partly arrays with n ($\geq$ k) parameter positions, k of which are open. The first of these cases, functions and procedures was felt to be no real problem, because with the new convention that identifiers in procedures are normally global it is very easy to define a new function with the correct number of open parameter positions from one with a larger number of such positions. This new function or procedure may then be used in the procedure statement.

The problem is much more acute in the case of arrays. In fact, within the European proposal there is no direct way of taking, say, one column of a matrix and use it as the actual input parameter to a procedure which works on a vector. The indirect ways are of course numerous: a) Rewrite the procedure in question in such a way that it takes in the complete array, but processes only part of it. b) Reassignment of the part of the array in question as a preparation for the procedure statement.

C. If bracket skeletons are omitted it becomes imperative that identifiers identify completely. This ties in with the advantage 5. of admitting functions and procedures without parameters.

D. The restriction of procedure declarations to admit the definition of only one procedure is based on three considerations:
D1. The desirability of defining several procedures in one declaration is removed when global parameters in procedures are admitted.
D2. Normal compound statements may be entered only through begin. Procedures should be similar.
D3. Since a function and a procedure cannot be mixed together in a single declaration it becomes possible to use the declarator function when declaring procedures of function-nature. Thus the concept function becomes complete in itself.

Draft for final report:

SEMANTICS OF TYPES AND ASSIGNMENT STATEMENTS.

Numbers and variables must be interpreted in the sense of numerical analysis, i.e. as entities defined inherently with only a finite accuracy. Similarly, the possibility of the occurrence of a finite deviation from the mathematically defined result in any arithmetic expression is explicitly understood. No exact arithmetic will be specified, however, and it is indeed understood that different hardware representations will realize arithmetic expressions differently. The control of the possible consequences of such differences must be carried out by the methods of numerical analysis. This control must be considered a part of the process to be described, and will therefore be described in terms of the language itself.

By means of a type declaration (section) a variable or function may be declared to belong to a certain class. Expressions containing variables or functions of different types must in general be assumed to have the type which mathematically speaking will embrace it in all cases. In special cases, however, the type of the expression may, for mathematical reasons, and in a given context, be assumed to belong to a more restricted type. If, and only if, this is the case may an expression be assigned to a variable of the more restricted type. Depending on the

characteristics of a specific hardware representation such an assignment may or may not give rise to special precautions, such as rounding, and this again may in the particular representation, cause that the assignment statement will yield a meaningful result even if the expression is not of the restricted type. It should be stressed, however, that this must be considered as an entirely incidental circumstance, which in no way changes the above strict rule, that only mathematically correct assignments are permitted in the language.

Draft for the final report:

PROCEDURE STATEMENTS.

General. A procedure statement serves to initiate (call for) the execution of a procedure, that is, a closed, selfcontained process with a fixed ordered set of input and output parameters, defined by a procedure declaration (cfr. section) valid in the level where the procedure statement occurs.

Syntactical rules.

<proc out param> ::= <identifier>|<subscr var>|<integer>
<output list> ::= <proc out param>|<output list>,<proc out param>
<output part> ::= <blank>|=:(<output list>)
<proc in param> ::= <identifier>|<exp>
<input list> ::= <proc in param>|<input list>,<proc in param>
<input part> ::= <blank>|(<input list>)
<name> ::= <identifier>
<procedure statement> ::= <name><input part><output part>
<procedure call> ::= <procedure statement>

Examples:

B(a/Q + v, b) =: (C[s-i, m] d, 18)
SR(v, u-z, t)
Isyp =: (p, epsilon, delta)
COMPILE

Here <name> is an identifier which is the name of some procedure, i.e. it appears in the heading of some procedure declaration (cfr. section) immediately following the delimiter procedure. <input list> and <output list> are ordered lists of entities defined in the level where the procedure statement occurs. The whole of the procedure statement must be identical in form with the part of the corresponding procedure declaration which appears between the delimiter procedure and the following ;. In this way there is defined a one-to-one correspondence between the identifiers or expressions in the input-output lists of the procedure statement and the formal identifiers in the input-output lists of the procedure heading. This one-to-one correspondence together with the input-output specifications given in the procedure heading (cfr. section) give complete information concerning the admissibility of parameters employed in any procedure call. Rules covering all admitted cases are given in the table below together with the corresponding semantic rules. In addition the following rules hold:

If a formal output parameter is identical to a formal input parameter, this identity must be preserved in the call. The correponding actual parameter as well as any other parameter entered both as input and output parameter in the procedure call must obviously meet the requirements of both input and output parameters.

Semantic rules. A procedure statement causes execution of the statement (usually compound) which is part of the procedure declaration (cfr. section). The execution, however, is effected as though all formal parameters listed in the procedure declaration heading were replaced, throughout the statement by actual parameters derived from the parameters in the corresponding positions in the procedure statement. In addition, the statements of the compound statement will in certain situations be supplemented by assignment statements inserted before the compound statement. The replacement rules and the statements executed are shown for all admissible cases below.

INPUT PARAMETERS.

Case 1.1.
Formal specification: typeF (f). (Special case: no specification for f).
Actual parameter: a.
Declaration for actual parameter: typeF (a).
Formal identifier will be replaced by: a.
Execution: Σ_p

Case 1.2.
Formal specification: typeF (f). (Special case: no specification for f).
Actual parameter: a.
Declaration for actual parameter: typeA (a). Quantities of typeA must form a subset of the quantities of typeF.
Formal identifier will be replaced by: g (a unique identifier of typeF).
Execution: g := a ; Σ_p . The assignment g:=a must be permitted, cfr. semantic of assignment statements, section .

Case 1.3.
Formal specification: typeF (f). (Special case: no specification for f).
Actual parameter: a[E1, ... , En].
Declaration for actual parameter: typeF array (a[E, ... ,E : E, ... , E].
Formal identifier will be replaced by: a[i, ... , k]. (i, ... , k are unique identifiers of type integer).
Execution: i := E1 ; .... ; k := En ; Σ_p . The assignments must be permitted ones, cfr. section .

Case 1.4.
Formal specification: typeF (f). (Special case: no specification for f).
Actual parameter: a[E1, ... , En].
Declaration for actual parameter: typeA array (a[E, ... ,E : E, ... , E].
Quantities of typeA must form a subset of the quantities of typeF.
Formal identifier will be replaced by: g (a unique identifier of typeF).
Execution: g := a[E1, ... , En] ;

Case 1.5.
Formal specification: typeF (f). (Special case: no specification for f).
Actual parameter: E (an expression in variables defined in the level where the procedure call is written).
Declaration for actual parameter: none.
Formal identifier will be replaced by: g (a unique identifier of typeF).
Execution: g := E ; Σ_p .

Case 2.1
Formal specification: typeF array (f[d, ... ,d : d, ... ,d]). (Special case: array (f[d, ... ,d : d, ... ,d])).
Actual parameter: a.
Declaration for actual parameter: typeF array (a[E, ... ,E : E, ... , E]. The number of subscripts and their bounds must be identical for f and a.
Formal identifier will be replaced by: a.
Execution: Σ_p .

Case 2.2. actual a[E ... E] (really open expression list in global variables and function)

Case 3.
Formal specification: typeF function (f(d, ... , d)). Special case: function (f(d, ... , d)).
Actual parameter: a.
Declaration for actual parameter: typeF function a(I, ... , I) .... This declaration must agree with the formal specification with respect to the number and nature of all parameters taken in the same order. If a uses global identifiers these must also be global to Σ_p .
Formal identifier will be replaced by: a.
Execution: Σ_p

Case 4.
Formal specification: procedure f(d, ... , d) =: (d, ... , d).
Actual parameter: a.
Declaration for actual parameter: procedure a ..... This declaration must agree with the formal specification with respect to the number and nature of all parameters taken in the same order. If a uses global identifiers these must also be global to Σ_p .
Formal identifier will be replaced by: a.
Execution: Σ_p

Case 5.
Formal specification: input f(d, ... , d).
Actual parameter: a.
Declaration for actual parameter: format a(.....). This format declaration must correspond to an input statement of the form given in the formal specificati
Formal identifier will be replaced by: a.
Execution: Σ_p

Case 6.
Formal specification: output f(d, ... ,d).
Actual parameter: a.
Declaration for actual parameter: format a(.....). This format declaration must correspond to an input statement of the form given in the formal specificati
Formal identifier will be replaced by: a.
Execution: Σ_p

OUTPUT PARAMETERS.

Case 7.1.
Same rules as for case 1.1.

Case 7.2.
Same rules as for case 1.3.

Case 8.
Same rules as for case 2.

Case 9.1
Formal specification: label (f).
Actual parameter: a. or <unsigned integer>
Declaration for actual parameter: none, but a must be a label accessible from the level of the procedure statement.
Formal identifier will be replaced by: a.
Execution: Σ_p

case 9.2. <unsigned integer>

Case 10.

Formal specification: switch (f:=(d, ... d)).
Actual parameter: a.
Declaration for actual parameter: switch a := (D, ... , D). The switch a must have the same number of positions as f. Variables occurring in any of the D's must be global to the procedure.
Formal identifier will be replaced by: a.
Execution: Σ_p

Return statements ....