DEPARTMENT OF SCIENTIFIC AND INDUSTRIAL RESEARCH

National Physical Laboratory

TEDDINGTON · MIDDLESEX

Please reply to the DIRECTOR *and quote our reference* *Telegrams :* Physics, Teddington *Telephone :* Molesey 1380, Ext.

OUR REF: Ma 8/29/01

YOUR REF:

25th January, 1960

Dear Peter,

... 1. I enclose a syntax of Algol 60 which is complete in as far as I understand the agreements reached on Saturday, 16th January in Paris.

2. With regard to your proposed amendments, I have incorporated the distinction between functions and procedures and have restricted functions to require at least one formal parameter, but I personally do not feel that either of these amendments is essential to avoid ambiguity. I do not have strong feelings either way, but your action is clearly consistent with having the fewest extensions of the language incorporated in Algol 60.

3. The following is the semantics of procedure statements as I understand it, based on my notes of 16th January.

Procedure statements

3.1 Parameter recognition.

An actual parameter appearing in a procedure statement may either be an identifier, or an expression other than a simple variable or label. This is a syntactical distinction recognizable by the translator.

An identifier represents either a variable, an array, a procedure or a function. When the procedure statement is executed there will be a unique quantity identified by each such identifier and therefore recognizable by the translator.

An expression may be arithmetic, Boolean or designational, and as a special case may be a subscripted variable. Since integer labels are not identifiers they appear in the category of designational expressions (an anomaly, but agreed!) and must be recognized by examination of the context of the corresponding formal parameters in the procedure compound.

3.2 Execution

If the procedure compound is not 'code' (outside the language) the procedure statement is executed as if it were replaced by the procedure compound and the following changes made in it immediately prior to the execution:

(1) Identifiers local to the compound which happen to be identical with actual parameters are systematically changed to avoid such conflicts.

(2) Actual parameters which are identifiers (including labels other than unsigned integers) are substituted for the corresponding formal parameters throughout the procedure compound. They are treated as global in the execution, and thus may already have initial values assigned to them.

(3) Actual parameters which are designational expressions other than just labels in the form of identifiers (but including integer labels) are evaluated and their values (labels or switch values with constant subscripts) are

/substituted

Mr. Peter Naur,
Regnecentralen,
Gl. Carlsbergvej 2,
Valby, DENMARK.

substituted for the corresponding formal parameters throughout the procedure compound, after first changing systematically any existing labels in the compound which happen to be identical with these. These substituted values are treated as global in the execution.

(4) Actual parameters which are arithmetic or Boolean subscripted variables have their subscript expressions evaluated. The subscripted variables with constant subscripts so obtained are substituted for the corresponding formal parameters throughout the procedure compound and are treated as global in the execution.

(5) Actual parameters which are arithmetic or Boolean expressions other than variables are evaluated and the values assigned to the corresponding formal parameters which are then treated as local in the execution.

4. The above agrees with a decision of 8.20 p.m. on Saturday night, that "in a procedure call the translator always transmits names", and accordingly makes no use of a name list in the procedure heading. The only ambiguity that seems to be possible is where the translator cannot tell from the formal context whether an unsigned integer actual parameter is to be treated as a (global) label (3) or as an arithmetic expression (5) to be assigned to the formal parameter as a (local) simple variable. A simple remedy would be to insist on such a label being followed by a colon when an actual parameter. In the example A (f) of your letter of 17th January case (2) applies and the function f is evaluated repeatedly whenever it occurs within the procedure compound of A. No notice need be taken by the translator at the time of the procedure call of how many formal parameters f has, if any. That question only arises when f is met within the procedure compound of A.

5. The semantics of function evaluation follows the same lines as above, and in view of the common provision of "failure exits" from function subroutines (so that they may involve labels as parameters) I see no need for making any distinction between functions and procedures. The syntax of the actual parameter list is the same in either case.

6. Concerning the syntax -

6.1 Apparently the procedure compound is not allowed to be a block (document 27). Was this intended?

✓ 6.2 The syntax of the logical negation sign was incorrect in document 30.

✓ 6.3 The syntax of 'designational expression' was incorrect in document 30.

✓ 6.4 The definition of 'compound tail' in document 30 was incomplete: it omitted 'else' and it allowed strings containing one or other but not both of ';' and 'end'.

Ney 6.5 The definitions in document 30 permitted basic statements to be labelled repeatedly but blocks and compounds only once. I understand that repeated labelling is permitted.

✓ 6.6 A delimiter 'block' is used in document 30 but never elsewhere and I do not remember admitting it. It is superfluous since it would in any case be immediately followed by 'begin <declarator>' which characterizes a block.

✓ 6.7 Although I believe we agreed to admit Backus' Item 24 of document 6 (string quotes and space) I have omitted them from the syntax in absence of any rules for their inclusion.

✓ 6.8 I find I have no record of the details agreed (if any) with regard to justaposition of declarators, so I have used common-sense (and 5.2.1 of document 5) in drawing up the syntax items 11, 18, 25 in which <type> and 'own' appear in combination with 'function' and 'array' and with each other.

/7.

7. With regard to document 1001 Rut the procedure x defined by

procedure x(a,b,c) begin c := a/b end

is evidently not restricted to integer parameters - that would be a matter for a program which used x - and I see no hardship in being unable to so restrict it.

In the case of

procedure x(a,b,c,d)L : begin ... A(b,c,d,e) ... end x;

the agreements as I understand them imply that in a particular use of x such as a call x(p,q + 1,r,s), the identifiers p,r,s are substituted for a,c,d respectively throughout L and treated as global, i.e. their signifiance as real, integer, arrays or what you will is defined by the accompanying program and has no bearing on the translation of this call of x. In this case the expression q + 1 is evaluated (the type of arithmetic depending on the type of q) and its current value assigned to the variable b, which will be treated as local. The inner call thus becomes A(b,r,s,e), in which all parameters are now simple identifiers and hence treated as global to the procedure compound of A after substitution. It is still quite immaterial that the signifiance of r and s is prescribed from outside the call of x.

With best wishes,

Yours sincerely,

Mike.

M. WOODGER

Copies to: Prof. Perlis
Prof. McCarthy
Mr. Backus
Mr. Green
Mr. Katz
Mr. Wegstein
Prof. Bauer
Prof. Rutishauser
Prof. Samelson
Mr. Vauquois
Prof. v. Wijngaarden

Syntax of Algol 60

1.<procedure declaration> ::= procedure<procedure heading>
<procedure compound>

2.<procedure compound> ;::= <compound statement>|<code>

3.<procedure heading> ::= <procedure identifier>
<formal parameter part>

4.<formal parameter part> ::= <empty>|(<formal parameter list>)

5.<formal parameter list> ::= <formal parameter>|
<formal parameter list><parameter delimiter><formal parameter>

6.<formal parameter> ::= <variable identifier>|<array identifier>|
<function identifier>|<procedure identifier>|<label identifier>|
<switch identifier>

7.<parameter delimiter> ::= ,|)<letter string>:(

8.<procedure identifier> ::= <identifier>

9.<label identifier> ::= <identifier>

10.<variable identifier> ::= <identifier>

11.<function declaration> ::= function<function heading>
<function compound>|<type>function<function heading>
<function compound>

12.<function compound> ::= <compound statement>|<code>

13.<function heading> ::= <function identifier>(<formal parameter list>)

14.<function identifier> ::= <identifier>

15.<switch declaration> ::= switch<switch identifier> := (<switch list>)

16.<switch list> ::= <designational e xpression>|
<switch list>,<designational expression>

17.<switch identifier> ::= <identifier>

18.<array declaration> ::= array(<array list>)|
<type>array(<array list>)|
own<type>array(<array list>)

19.<array list> ::= <array segment>|<array list>,<array segment>

20.<array segment> ::= <array identifier>[<lower upper bound list>]|
<array identifier>,<array segment>

21.<lower upper bound list> ::= <lower bound>:<upperbound>|
<lower upperbound list>,<lower bound>:<upper bound>

22.<lower bound> ::= <arithmetic expression>

23.<upper bound> ::= <arithmetic expression>

24.<array identifier> ::= <identifier>

25.<type declaration> ::= <type>(<type list>)|own<type>(<type list>)

26. <type> ::= integer|Boolean

27. <type list> ::= <simple variable>|<type list>,<simple variable>

28. <own declaration> ::= own(<identifier list>)

29. <real declaration> ::= real(<identifier list>)

30. <identifier list> ::= <identifier>|<identifier list>,<identifier>

---

31. <declaration list> ::= <declaration>|< declaration list>;<declaration>

32. <declaration> ::= <real declaration>|<own declaration>|
<type declaration>|<array declaration>|<switch declaration>|
<function declaration>|<procedure declaration>

---

33. <procedure statement> ::= <procedure identifier>
<actual parameter part>

34. <actual parameter part> ::= <empty>|(<actual parameter list>)

35. <actual parameter list> ::= <actual parameter>|
<actual parameter list> <parameter delimiter> <actual parameter>

36. <actual parameter> ::= <variable identifier>|<expression>|
<array identifier>|<procedure identifier>|
<function identifier>|<designational expression>

---

37. <dummy statement> ::= <empty>

38. <assignment statement> ::= <arith assignment statement>|
<boolean assignment statement>

39. <arith assignment statement> ::= <variable> := <arith expression>|
<variable> := <arith assignment statement>

40. <boolean assignment statement> ::= <variable> := <boolean expression>|
<variable> := <boolean assignment statement>

---

41. <go to statement> ::= go to<designational expression>

42. <for statement> ::= for<variable> := <arith expression list>do
<statement>

43. <arith expression list> ::= <arith expr list elem>|
<arith expression list>,<arith expr list elem>

44. <arith expr list elem> ::= <arithmetic expression>|
<arith expression>step<arith expression>until<arith expression>|
<arith expression>while<boolean expression>

---

45. <statement> ::= <conditional stat>|
<conditional stat>else<unconditional statement>|
<unconditional statement>

46. <conditional stat> ::= <simple conditional stat>|
<conditional stat>else<simple conditional stat>

47. <simple conditional stat> ::= if<boolean expression>then
<unconditional statement>

48.<unconditional statement> ::= <basic statement>|
<compound statement>|<block>

49.<block> ::= <unlabelled block>|<label>:<block>

50.<unlabelled block> ::= <for statement>|<simple block>

51.<simple block> ::= <block head> <compound tail>

52.<block head> ::= begin<declaration>|<block head>;<declaration>

53.<compound statement> ::= <unlabelled compound>|
<label>:<compound statement>

54.<unlabelled compound> ::= begin<compound tail>

55.<compound tail> ::= <compound end>|
<compound end> <any string not containg ; or end or else>

56.<compound end> ::= <statement> end|<statement>;<compound end>

57.<basic statement> ::= <unlabelled basic statement>|
<label>:<basic statement>

58.<unlabelled basic statement> ::= <assignment statement>|
<go to statement>|<procedure statement>|<dummy statement>

59.<designational expression> ::= <conditional des expr>|
<conditional des expr>else<d prime>|<d prime>

60,<conditional des expr> ::= <simple con des expr>|<
<conditional des expr>else<simple con des expr>

61.<simple con des expr> ::= if<boolean expression>then<d prime>

62.<d prime> ::= <label>|<switch value>

63.<label> ::= <identifier>|<unsigned integer>

64.<switch value> ::= <switch identifier>[<subscript expression>]

65.<subscript expression> ::= <arithmetic expression>

66.<boolean expression> ::= <conditional boolean>|
<conditional boolean>else<minor boolean>|<minor boolean>

67.<conditional boolean> ::= <simple con boolean>|
<conditional boolean>else<simple con boolean>

68.<simple con boolean> ::= if<boolean expression>then<minor boolean>

69.<minor boolean> ::= <implication boolean> |
<minor boolean> ≡ <implication boolean>

70.<implication boolean> ::= <b term>|<implication boolean> ⊃ <b term>

71.<b term> ::= <b factor>|<b term> ∨ <b factor>

72.<b factor> ::= <b prime>|<b factor> ∧ <b prime>

73.<b prime> ::= (<boolean expression>)|<logical value>|<relation>|
<variable>|<function value>| ¬ <b prime>

↓

¬¬¬¬¬ q

74.<logical value> ::= true|false

75.<relation> ::= <prime expr> <relational operator> <prime expr>

76.<relational operator> ::= <|≤|=|≥|>|≠

77.<expression> ::= <arithmetic expression>|<boolean expression>

78.<arithmetic expression> ::= <conditional arith expr>|
<conditional arith expr>else<term sum>|<term sum>

79.<conditional arith expr> ::= <simple con arith>|
<conditional arith expr>else<simple con arith>

80.<simple con arith> ::= if<boolean expression>then<term sum>

81.<term sum> ::= <term>|<add op> <term>|<term sum> <add op> <term>

82.<term> ::= <factor>|<term> <mult op> <factor>

83.<factor> ::= <prime expr>|<factor>↑<prime expr>

84.<prime expr> ::= (<arithmetic expression>)|<unsigned number>|
<variable>|< function value>

85.<add op> ::= +|-

86.<mult op> ::= ×|/|÷

87.<function value> ::= <function identifier>(<actual parameter list>)

88.<variable> ::= <simple variable>|<subscripted variable>

89.<subscripted variable> ::= <array identifier>[<subscript list>]

90.<subscript list> ::= <subscript expression>|
<subscript list>,<subscript expression>

91.<simple variable> ::= <identifier>

92.<number> ::= <unsigned number>|<add op> <unsigned number>

93.<unsigned number> ::= <decimal number>|<exponent part>|
<decimal number> <exponent part>

94.<decimal number> ::= <unsigned integer>|<decimal fraction>|
<unsigned integer> <decimal fraction>

95.<exponent part> ::= $_{10}$<integer>

96.<decimal fraction> ::= .<unsigned integer>

97.<integer> ::= <unsigned integer>|<add op> <unsigned integer>

98.<letter string> ::= <letter>|<letter string> <letter>

99.<identifier> ::= <letter>|<identifier> <letter>|<identifier> <digit>

100.<unsigned integer> ::= <digit>|<unsigned integer> <digit>

101.<basic symbol> ::= <letter>|<digit>|<logical value>|<delimiter>

102.<letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

103.<digit> ::= 0|1|2|3|4|5|6|7|8|9

104.<delimiter> ::= <operator>|<separator>|<bracket>|<declarator>

105.<operator> ::= <arithmetic operator>|<relational operator>|
<logical operator>|<sequential operator>

106.<arithmetic operator> ::= <add op>|<mult op>|↑

107.<logical operator> ::= ¬ |∧|∨|⊃|≡

108.<sequential operator> ::= go|to|for|step|until|while|do|if|then|else

109.<separator> ::= ,|;|:|:=|$_{10}$ |comment

110.<bracket> ::= (|)|[|]|begin|end

111.<declarator> ::= procedure|function|array|switch|<type>|own|real

112.<empty> ::= <the null string of symbols>

113.; is syntactically equivalent to
comment<any string not containing ;>;

25th January, 1960
MW/PL (15)