6. Statements

<statement> ::= <ALGOL unconditional statement>|
<conditional statement>|
<for statement>|
<connection statement>

<unlabelled basic statement> ::= <assignment statement>|
<go to statement>|
<dummy statement>|
<procedure statement>|
<activation statement>|
<object generator>

<conditional statement> ::= <ALGOL conditional statement>|
<if clause><connection statement>

For <connection statement> see section 7.2.
For <activation statement> see section 14.2.3.

6.1 Assignment statements

6.1.1 Syntax

<assignment statement> ::= <value assignment>|
<reference assignment>

<value left part> ::= <variable>|
<procedure identifier>|
<simple text expression>

<value right part> ::= <value expression>|
<text value>|
<value assignment>

<value assignment>::=
<value left part> := <value right part>

<reference left part> ::= <variable>|
<procedure identifier>

<reference right part> ::= <reference expression>|
<reference assignment>

<reference assignment> ::=
<reference left part> :- <reference right part>

6.1.2 Semantics

The operator ":=" (read: "becomes") indicates the assignment of a value to the value type variable or value type procedure identifier which is the left part of the value assignment or the assignment of a text value to the text object referenced by the left part.

The operator ":-" (read: "denotes") indicates the assignment of a reference to the reference type variable or reference type procedure identifier which is the left part of the reference assignment.

A procedure identifier in this context designates a memory device local to the procedure instance. This memory device is initialized upon procedure entry according to section 3.2.4.

The value or reference assigned is a suitably transformed representation of the one obtained by evaluating the right part of the assignment. If the right part is itself an assignment, the value or reference obtained is a copy of that of its constituent left part after that assignment operation has been completed.

Any expression which is, or is part of, the left part of an assignment is evaluated prior to the evaluation of the right part.

For a detailed description of the text value assignment, see section 10.6. There is no value assignment operation for objects.

The type of the value or reference obtained by evaluating the right part, must coincide with the type of the left part, with the exceptions mentioned in the following sections.

If the left part of an assignment is a formal parameter, and the type of the corresponding actual parameter does not coincide with that of the formal specification, then the assignment operation is carried out in two steps.

1) An assignment is made to a fictitious variable of the type specified for the formal parameter.

2) An assignment statement is executed whose left part is the actual parameter and whose right part is the fictitious variable.

The value or reference obtained by evaluating the assignment is, in this case, that of the fictitious variable.

For text reference assignment see section 10.5.

6.1.2.1 Arithmetic value assignment

In accordance with ALGOL 60, any arithmetic value may be assigned to a left part of type real or integer. If necessary, an appropriate transfer function is invoked.

Example:

Consider the statement (not a legal one in ALGOL 60):

```
X := i := Y := F := 3.14
```

where X and Y are real variables, i is an integer variable, and F is a formal parameter called by

name and specified real. If the actual parameter for F is a real variable, then X, i, Y and F are given the values 3,3,3.14 and 3.14 respectively. If the actual parameter is an integer variable, the respective values will be 3,3,3.14 and 3.

6.1.2.2 Object reference assignment

Let the left part of an object reference assignment be qualified by the class Cl, and let the right part be qualified by Cr. If the right part is itself a reference assignment, Cr is defined as the qualification of its constituent left part. Let V be the value obtained by evaluating the right part. The legality and effect of the reference assignment depend on relationships between Cr, Cl and V.

Case 1. Cl is of the class Cr or outer to Cr:
The reference assignment is legal and the assignment operation is carried out.

Case 2. Cl is inner to Cr:
The reference assignment is legal. The assignment operation is carried out if V is none or is an object belonging to the cåass Cl or innracters

class Cl or a class inner to Cl. If not, the execution of the reference assignment constitutes a run time error.

Case 3. Cl and Cr satisfy neither of the above relations:
The reference assignment is illegal.

Similar rules apply to reference assignments implicit in for clauses and the transmission of parameters.

Example 1:

Let "Gauss" be the class declared in the example of the section 2.2.

```
ref (Gauss) G5, G10;
    G5 :- new Gauss(5);  G10 :- new Gauss(10);
```

The values of G5 and G10 are now Gauss objects.
See also example 1 of section 7.1.2.

Example 2:

Let "point" and "polar" be the classes declared in the example of section 2.2.2.

```
ref (point) P1, P2; ref (polar) P3;
    P1 :- new polar (3,4); P2 :- new point (5,6);
```

Now the statement "P3 :- P1" assigns to P3 a reference to the "polar" object which is the value of P1. The statement "P3 :- P2" would cause a run time error.

6.2 For statements

6.2.1 Syntax

```
<controlled variable> ::= <simple variable>
<controlled statement> ::= <statement>
<for statement> ::= <for clause><controlled statement>|
                    <label> : <for statement>
<for clause> ::= for <controlled variable>
                     <for right part> do
<for right part> ::= :=<value for list>|
                     :-<object for list>
<value for list> ::= <value for list element>|
                     <value for list>,
                        <value for list element>
```

<object for list> ::= <object for list element>|
<object for list>,
<object for list element>
<value for list element> ::= <value expression>|
<arithmetic expression> step <arithmetic expression> until <arithmetic expression>|
<value expression> while <Boolean expression>
<object for list element> ::= <object expression>|
<object expression> while <Boolean expression>

6.2.2 Semantics

A for clause causes the controlled statement to be executed repeatedly zero or more times. Each execution of the controlled statement is preceded by an assignment to the controlled variable and a test to determine whether this particular for list element is exhausted.

Assignments may change the value of the controlled variable during execution of the controlled statement.

6.2.3 For list elements

The for list elements are considered in the order in which they are written. When one for list element is exhausted, control proceeds to the next, until the last for list element in the list has been exhausted. Execution then continues after the controlled statement.

The effect of each type of for list element is defined below using the following notation:

C: controlled variable
V: value expression
O: object expression
A: arithmetic expression
B: Boolean expression
S: controlled statement

The effect of the occurrence of expressions as for list elements may be established by textual replacement in the definitions.

α, β, σ are different identifiers which are not used elsewhere in the program. σ identifies a non-local simple variable of the same type as A_2.

1. V

```
C:= V;
S;
next for list element
```

2. A_1 step A_2 until A_3

```
C := A1;
σ := A2;
α: if σ ×(C-A3) > 0 then go to β;
S;
σ := A2;
C := C + σ;
go to α;
β: next for list element
```

3. V while B

```
α: C := V;
   if ¬ B then go to β;
   S;
   go to α;
β: next for list element
```

4. 0

```
C :- 0;
S;
next for list element
```

5. O while B

```
α: C :- O;
   if ¬ B then go to β;
   S;
   go to α;
β: next for list element
```

6.2.4 The controlled variable

The semantics of this section (6.2) is valid when the controlled variable is a simple variable which is not a formal parameter called by name, or a procedure identifier.

The cases of formal parameter called by name, procedure identifier, subscripted variable and remote identifier are presently under study by a Technical Committee appointed by the SIMULA Standards Group.

To be valid, all for list elements in a for-statement (defined by textual substitution, section 6.2.3) must be semantically and syntactically valid.

In particular each implied reference assignment in cases 4 and 5 of section 6.2.3 is subject to the rules of section 6.1.2.2.

6.2.5 The value of the controlled variable upon exit

Upon exit from the for statement, the controlled variable will have the value given to it by the last (explicit or implicit) assignment operation.

6.2.6 Labels local to the controlled statement

The controlled statement always acts as if it were a block. Hence, labels on or defined within the controlled statement may not be accessed from without the controlled statement.

6.3 Prefixed blocks

6.3.1 Syntax

```
<block> ::= <ALGOL block>|
            <prefixed block>
<block prefix> ::=
    <class identifier><actual parameter part>
<main block> ::= <unlabelled block>|
                 <unlabelled compound>
<unlabelled prefixed block> ::=
                <block prefix><main block>
<prefixed block> ::= <unlabelled prefixed block>|
                     <label>:<prefixed block>
```

6.3.2 Semantics

An instance of a prefixed block is a compound object whose prefix part is an object of the class identified by the block prefix, and whose main part is an instance of the main block. The formal parameters of the former are initialized as indicated by the actual parameters of the block prefix. The concatenation is defined by rules similar to those of section 2.2.2.

The following restrictions must be observed:

1) A class in which reference is made to the class itself through use of "this", is an illegal block prefix.

2) The class identifier of a block prefix must refer to a class local to the smallest block enclosing the prefixed block. If that class identifier is that of a system class, it refers to a fictitious declaration of that system class occurring in the block head of the smallest enclosing block.

An instance of a prefixed block is a detached object (cf. section 9). A program is enclosed in a prefixed block (cf. section 11) and is therefore detached.

Example:

Let "hashing" be the class declared in the example of section 2.2.3. Then within the prefixed block,

```
hashing (64) begin integer procedure hash(T); value T;
                   text T; ............;
                ..............
             end
```

a "lookup" procedure is available which makes use of the "hash" procedure declared within the main block.