Proposals for Consideration by the SIMULA 67 Common Base Conference, June 1967

by

Ole-Johan Dahl and Kristen Nygaard.

1. Introduction

Since the "SIMULA 67 Common Base Proposal", [4] was published, the authors have been working on the problem of in-line declarations. It turns out that a more consistent language may be obtained by adopting another approach than given in [4].

There has been a desire to transfer the <type> procedure concept over to the class declaration, preferably allowing generalizations to types other than the basic ones. A solution to this problem is presented in this paper.

Both these considerations make it necessary to be able to define operations on non-basic types in a flexible way.

At the IFIP Working Conference on Simulation Languages the question of side-effects to operations on objects belonging to basic types was much discussed. It turns out that this and the above problems are connected, and a unified solution is given in this paper.

The solution is inspired by the GPL language designed by J.V. Garwick. In order to avoid excessive cross-referencing, this paper also repeats the relevant parts of the "SIMULA 67 Common Base Proposal".

SIMULA 67 Common Base is a true extension of ALGOL 60 (1), except that the own concept is deleted, integer labels are not allowed, and parameter specifications are required. The extensions are defined by reference to the documents (2) and (3).

2. Type Declarations

2.1 Syntax

<basic type>: := real|integer|Boolean|string| ref|ref<qualification>
<qualification>: :=<type>

<type>: := <basic type> | <class id.>
<in-line object>: := <identifier><actual parameter part>
<type declaration>: := <type><in-line object> | <type declaration>,
<in-line object>.

2.2 Semantics

In addition to the basic types, composite types can be defined by class declarations. A class identifier can be used as a type declarator, provided that the class body contains no assignment of a <local reference>. The classes "link", "set", "process" and "SIMULA" are examples of classes which may not be used as type declarators.

Variables of the five basic types are initialized by implied class bodies to the respective values "0", "0", "false", blanks and "none".

The declaration of an in-line object causes the execution of the corresponding class body. The generated object becomes an integral part of the data structure local to the block in which the declaration occurs. The actual parameters of in-line objects are specified separately for each declared identifier.

3. Expressions

3.1 Syntax

<expression>: := <arithmetic expr.>|<Boolean expr.>|<string expr.>|
<reference expr.>|<reference expr.>.
<function designator>: := <procedure id.><actual param.part>|
<class id.><actual param.part>.

3.2 Semantics

An expression is a rule for computing a value (which is an object) of either a basic or composite type. In the latter case the value belongs to a declared class. There is a distinction between a value which is an object and a value of type ref referring to the same object.

A reference expression followed by a "dot" (.) denotes the object referenced by the reference value. If the reference value is none the result is undefined.

A function designator referring a class declaration has a value which is a generated object belonging to the class. The class body is executed as part of the evaluation.

3.2.1 Parameter Correspondence

The type of an actual parameter should be a class included in the one specified for the corresponding formal parameter. If the latter is specified to type "ref (C)" the actual parameter should be of type "ref (D)" (or any subclass of ref (D), where D is included in C.

3.3. Operations on Composite Values

Operations on a non-basic object can be defined by procedure declarations local to the object. There is a one to one correspondence between operators and identifiers, according to the following list:

:=	becomes
+	plus
-	minus
x	mult
/	div
÷	divint
↑	power
=	equal
≠	unequal
>	greater
≥	greateq
<	less
≤	lesseq
∧	and
∨	or
⊃	implies
≡	equivalent

The basic types are thought of as classes containing local declarations of the relevant procedures from the above list. Each operator is defined by source language transformations of the following type:

a op b ⟶ a' op(b)

where "op" is an operator and "op" is the corresponding procedure identifier. The apostrophe (') is an operator which gives access to quantities local to the value of the preceding expression, see next section, amendment to (3), 5.1.

4. Classes and References

SIMULA 67 should include all concepts described in (3), sections 2-6 and 8. The following amendments should be made:

To 2.2 Addition to first paragraph, p.4. "..... provided that the corresponding identifiers have not been redeclared within the main part. Redeclaration of an attribute **identifier** will supercede within the main part the declaration given in the prefix, but will have no effect for its meaning within the prefix body".

To 3.1.1 Should be substituted by the two first definitions in 2.1 in this paper.

To 3.2.1 Replaced by

<simple ref.expr.>::=none|<variable 1>|<function designator>|
<object designator>|<local reference>|
<remote reference>
<ref.expr.>::=<simple ref.expr.>|if<Boolean expr.> then
<simple ref.expr.> else <ref.expr.>
<object designator>::=new<class id.><actual parameter part>
<local reference>::=this<type>
<remote reference<::=<reference expr.>.this<type>

To 3.2.2 Add new sub-section:

3.2.2.4 Remote Reference

A remote reference "E.this C" is a reference expression, whose value is that of E. C serves as the qualification of the expression. It will supercede any qualification inherent in E. A runtime error will occur if the value of E is none, or if the referenced object does not belong to a class included in C.

To 4.2.1 Add as new alternative:
<reference expr.><class op.><reference expr.>
where
<class op.>::=>|≥|<|≤|≡|≠

To 4.2.2 Add at the end of the section:

A <class op.> compares the class memberships of the operands. E.g. "X>Y" is true if the class of Y is a subclass of and different from the class of X. The operators "=" and "≠" compare identity of objects.

To 5. Replace main section by:

An attribute of an object is identified completely by the following items of information:

1) An object identification.

2) a <class id.> specifying a class, which includes that of the object, and

3) the <identifier> of an attribute declared for objects of the stated class.

The class identification, item 2, must be available at compile time, in order to obtain run time efficiency.

For a local reference to an attribute, i.e. a reference from within the class body, items 1 and 2 are defined implicitly. Item 1 is the current instance (i.e. object), and item 2 is the class identifier of the class declaration.

Non-local (remote) referencing is either through component identifiers or through connection. The former is an adaptation of the technique proposed in (2), the latter corresponds to the connection mechanism of SIMULA (1).

To 5.1 Replace whole subsection by

"5.1 Components

5.1.1 Syntax

<component identifier>::=<expression>'<identifier>
<identifier 1>::=<identifier>|<component identifier>

Replace the meta-variable <identifier> by <identifier 1> at appropriate places of the ALGOL syntax. The above syntax is simplified by omitting the apostrophe if the <expression> is a reference expression followed by a dot.

5.1.2 Semantics

A componenet identifier identifies an attribute of an individual object. Item 2 above is defined by the type of the <expression> whose value is item 1. (If the expression is a reference expression followed by a dot, the type is defined by the qualification).

To 5.2.1 Replace "otherwise<connection block 2>"
by "otherwise<statement>"

To 5.2.2 Replace last sentence by:

"The result of entering a <connection block 2> is undefined if the reference expression has the value none".

To 6.2 Replace by:

6.2 Non-existing object

The application of the dot operator to the reference value none should cause an error printout and program termination. Also the entering of a <connection block 2> should cause a runtime error if the reference expression of the connection statement has the value none.

5. The SIMULA Class

The symbol SIMULA is thought of as a system defined class. In addition to introducing a number of procedure and class declarations (see below) the SIMULA prefix also introduces special syntax for scheduling statements.

The semantic contents of the process concept is as defined in 2, with the following modifications:

1) A process is an object whith an associated reference value. There is no implied element concept as in (2).

2) The declarator activity is replaced by the construction "process class".

A formal definition of the class "process" can be given in terms of the extended class concept of (3), as outlined in (3),8.3. The process class shall have the prefix "link" (see below).

6. Elements and Sets

Elements and sets will be implemented as classes declared in the SIMULA class:

```
1.  class linkage; begin ref (linkage) succ, prede; end;

2.  linkage class link;

    begin procedure out;
          if succ ≠ none then begin
          succ.prede := prede; pred.succ := succ;
          succ := prede := none end;
          procedure follow (X); value X; ref (linkage)X;
          begin out;
          if X ≠ none ∧ X.succ ≠ none then begin
          prede := X; succ := X.succ;
          X.succ.prede := X.succ := this linkage end end;
          procedure precede (X); value X; ref (linkage)X;
          begin out;
          if X ≠ none ∧ X.succ ≠ none then begin
          succ := X; prede := X.prede;
          X.prede.succ := X.prede := this linkage end end;
          procedure into (S); value S; ref(set)S;
          begin out; succ := S; prede := S.prede;
          S.prede.succ := S.prede := this linkage end;
          ref (link) procedure suc; suc := succ;
          ref (link) procedure pred; pred := prede;
      end link;

3.  link class element (object); ref object;;

4.  linkage class set;
    begin ref (link) procedure first; first := succ;

          ref (link) procedure last; last := prede;

          Boolean procedure empty; empty := succ = prede;

          integer procedure cardinal;
          begin integer i; ref (link) X; i := 0;
          for X := first, X.suc while X is link do i := i+1;
          cardinal := i end;

          procedure clear:
          begin ref (link) X;
          for X := first while X is link do X. out end;
```

```
ref (element) procedure member (X); value X; ref X;
begin ref (element) Y; member := none;
for Y := first, Y.suc while Y is link do
if Y.object = X then begin member := Y; go to fin end;
fin: end;

succ := prede := this linkage
end set;
```

It is important to provide security in the operation on sets. The following ad hoc rule should be applied to variables of type ref(set): The value none is illegal, and they are initialized to the value "new set". This rule will reduce the amount of run-time checks on this type.

5. The following SIMULA I procedures will not be implemented:

proc(X): substituted by the generative expression element (X)

head(S): no meaning within SIMULA 67

suc(X): substituted by "suc" local to "link"

pred(X): substituted by "pred" local to "link"

same(X,Y): of little interest since elements are not a basic part of SIMULA 67.

similar(X,Y), X=Y and X ≠ Y will be substituted by Boolean expressions.

pred(X,Y): substituted by "precede" local to "link"

remove(X): substituted by "out" local to "link"

first(S): substituted by "first" local to "set"

last(S): substituted by "last" local to "set"

successor(n,X): omitted since it has not been proved useful in SIMULA I.

number(n,S): as for successor (n,X)

member(X,S): substituted by "member" local to "set"

exist(X): substituted by X = none, since "sethead" does not appear in SIMULA 67

empty(S): substituted by "empty" local to "set"

ordinal(X): omitted since it has not been proved useful in SIMULA I. Also inefficient in execution. May be user defined if necessary.

cardinal(S): substituted by "cardinal" local to "set"

precede(X,Y): substituted by "precede" local to "link"

follow(X,Y): substituted by "follow" local to "link"

transfer(X,S): substituted by "into" local to "link"

include(X,S): substituted by "into" local to "link"

clear(S): substituted by "clear" local to "set"

The variables "succ" and "prede" local to "linkage" will not be available to the user.

7. Sequencing

Chapter 4 in the SIMULA I manual will apply inside the SIMULA class with some exceptions:

The event notices will refer directly to processes, not through elements.

The statement go to L will be undefined if the label is not local to the current process.

In all sequencing statements the element expression in (2) are replaced by ref (process) expressions.

8. Random Drawing

SIMULA 67 specifications correspond to Chapter 7 in SIMULA I manual, 4th edition.

9. Data Analysis

SIMULA 67 specifications correspond to Chapter 8 in SIMULA I manual, 4th edition, except for the "hprint" procedure which should not be considered as a part of SIMULA 67 Common Base.

10. Extensions

The following extensions should be discussed for inclusion in the SIMULA 67 Common Base.

10.1 Sequencing

Simple basic statements for quasi-parallel sequencing are described in (3),8.3. For security reasons the statements resume(X) and goto(X) should not be available within a SIMULA block.

10.2 Virtual Quantities

The parameters to a class declaration are called by value. Call by name is difficult to implement with full security and good efficiency. The main difficulty is concerned with the definition of the dynamic scope of the actual parameter corresponding to the formal name parameter. It is felt that the cost of an unrestricted call by name mechanism would in general be out of proportion to its gain, and that it would represent an invitation to misuse computers.

The virtual quantities described below represent another approach to call by name in class declarations The mechanism provides access at one prefix level of the prefix sequence of an object to quantities declared local to the object at lower levels.

10.2.1 Syntax

<class declaration>::=<prefix><class declarator><class id.>
<formal parameter part>;
<specification part><virtual part>
<class body>

<virtual part>::=<empty>|virtual:<specification part>

10.2.2 Semantics

The identifiers of a <virtual part> should not otherwise occur in the heading or in the block head of the class body. Let A_1, ..., A_n be the prefix sequence of A_o and let X be an identifier occuring in the <virtual part> of A_i. If X identifies a parameter of A_j pr a quantity declared local to the body of A_j, $j<i$, then for an object of class A_o identity is established between the virtual quanitity X and the quantity X local to A_j.

If there is no A_j, $j<i$, for which X is local, a reference to the virtual quantity X of the object constitutes a run time error.

REFERENCES

1. P. Naur, ed.: "Revised Report on the Algorithmic Language ALGOL 60".

2. O-J. Dahl and K. Nygaard: "SIMULA - A language for programming and description of discrete event systems. Introduction and users' manual". Norwegian Computing Center, 4th edition, Oslo, April 1967.

3. O-J. Dahl and K. Nygaard: "Class and sub-class declarations". Paper for the IFIP Working Conference in Oslo, May 1967. Norwegian Computing Center, March 1967.

4. O-J. Dahl and K. Nygaard: "SIMULA 67 Common Base Proposal". Norwegian Computing Center, May 1967.