2. Class declarations

2.1 Syntax

```
<declaration> ::= <ALGOL declaration>|
                  <class declaration>|
                  <external declaration>
<class identifier> ::= <identifier>
<prefix> ::= <empty>|
             <class identifier>
<virtual part> ::= <empty>|
                   virtual: <specification part>
<class body> ::= <statement>|
                 <split body>
<initial operations> ::= begin|
                         <blockhead>;|
                         <initial operations><statement>;
<final operations> ::= end|
                       ; <compound tail>
<split body> ::= <initial operations>
                 inner <final operations>
<class declaration> ::= <prefix><main part>
<main part> ::= class <class identifier>
                <formal parameter part>;
                <value part><specification part>
                <virtual part><class body>
```

2.2 Semantics

A class declaration serves to define the class associated with a class identifier. The class consists of "objects" each of which is a dynamic instance of the class body.

An object is generated as the result of evaluating an object generator, which is the analogy of the "call" of a function designator, see section 4.3.2.2.

A class body always acts like a block. If it takes the form of a statement which is not an unlabelled block, the class body is identified with a block of the form

begin; S end

when S is the textual body. A split body acts as a block in which the symbol "inner" represents a dummy statement.

For a given object the formal parameters, the quantities specified in the virtual part, and the quantities declared local to the class body are called the "attributes" of the object. A declaration or specification of an attribute is called an "attribute definition".

Specification (in the specification part) is necessary for each formal parameter. The parameters are treated as variables local to the class body. They are initialized according to the rules of parameter transmission, (see section 8.2). Call by name is not available for parameters of class declarations. The following specifiers are accepted:

| <type>, array, and <type> array.

Attributes defined in the virtual part are called "virtual quantities". They do not occur in the formal parameter list. The virtual quantities have some properties which resemble formal parameters called by name. However, for a given object the environment of the corresponding "actual parameters" is the object itself, rather than that of the generating call. See section 2.2.3.

Identifier conflicts between formal parameters and other attributes defined in a class declaration are illegal.

The declaration of an array attribute may in a constituent subscript bound expression make reference to the formal parameters of the class declaration.

Example:

The following class declaration expresses the notion of "n-point Gauss integration" as an aggregated concept.

```
class Gauss (n); integer n;
   begin array W,X[1:n];
      real procedure integral(F,a,b); real procedure F;
                                      real a,b;
        begin real sum, range; integer i;
           range := (b-a) ×0.5;
           for i := 1 step 1 until n do
           sum := sum + F(a+range×(X[i]+1))×W[i];
           integral := range × sum;
        end integral;
      comment compute the values of the elements of
              W and X as functions of n;
      ..........
   end Gauss;
```

The optimum weights W and abcissae X can be computed as functions of n. By making the algorithm part of the class body, the evaluation and assignment of these values can be performed at the time of object generation. Several "Gauss" objects with different values of n may co-exist. Each object has a local procedure "integral" for the evaluation of the corresponding n-point formula. See also examples of section 6.1.2.2 and section 7.1.2.

2.2.1 Subclasses

A class declaration with the prefix "C" and the class identifier "D" defines a subclass D of the class C. An object belonging to the subclass consists of a "prefix part", which is itself an object of the class C, and a "main part" described by the main part of the class declaration. The two parts are "concatenated" to form one compound object. The class C may itself have a prefix.

Let C_1, C_2, ..., C_n be classes such that C_1 has no prefix and C_k has the prefix C_{k-1} ($k = 2, 3, ..., n$). Then C_1, C_2, ..., C_{k-1} is called the "prefix sequence" of C_k ($k = 2, 3, ..., n$). The subscript k of C_k ($k = 1, 2, ..., n$) is called the "prefix level" of the class. C_i is said to "include" C_j if $i \leq j$, and C_i is called a "subclass" of C_j if $i > j$ ($i, j = 1, 2, ..., n$). The prefix level of a class D is said to be "inner" to that of a class C if D is a subclass of C, and "outer" to that of C if C is a subclass of D. The figure 2.1 depicts a class hierarchy consisting of five classes, A, B, C, D and E:

```
  class A .......;
A class B .......;
B class C .......;
B class D .......;
A class E .......;
```

A capital letter denotes a class. The corresponding lower case letter represents the attributes of the main part of an object belonging to that class. In an implementation of the language, the object structures shown in Fig. 2.2 may indicate the allocation in memory of the values of those attributes which are simple variables.

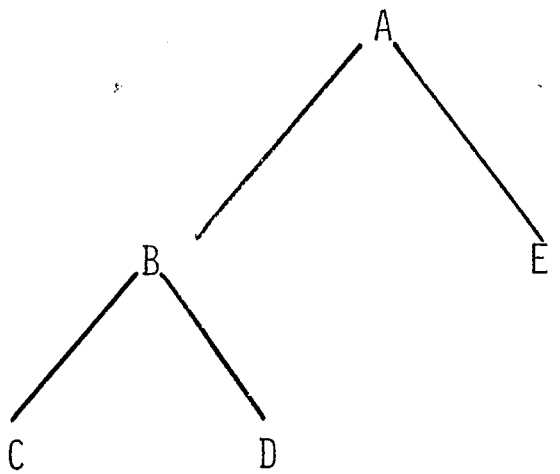

Fig. 2.1

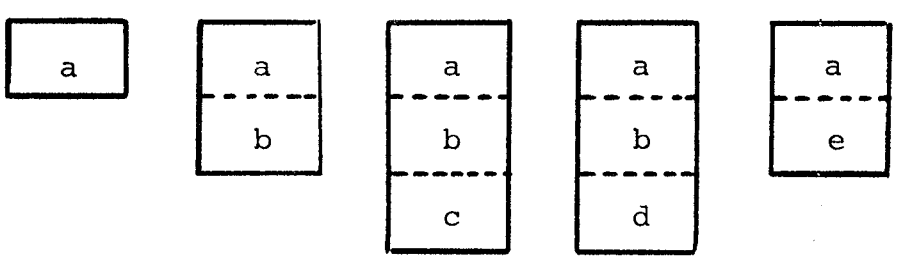

Fig. 2.2

The following restrictions must be observed in the use of prefixes:

1) A class must not occur in its own prefix sequence.

2) A class can be used as prefix only at the block level at which it is declared. A system class is considered to be declared in the smallest block enclosing its first textual occurrence. An implementation may restrict the number of different block levels at which such prefixes may be used. See sections 11, 14 and 15.

2.2.2 Concatenation

Let C_n be a class with the prefix sequence C_1, C_2, ..., C_{n-1}, and let X be an object belonging to C_n. Informally, the concatenation mechanism has the following consequences.

1) X has a set of attributes which is the union of those defined in C_1, C_2, ..., C_n. An attribute defined in C_k ($1 \leq k \leq n$) is said to be defined at prefix level k.

2) X has an "operation rule" consisting of statements from the bodies of these classes in a prescribed order. A statement from C_k is said to belong to prefix level k of X.

3) A statement at prefix level k of X has access to all attributes of X defined at prefix levels equal to or outer to k, but not directly to attributes "hidden" by conflicting definitions at levels $\leq k$. These "hidden"attributes may be accessed through use of procedures or this).

4) A statement at prefix level k of X has no immediate access to attributes of X defined at prefix levels inner to k, except through virtual quantities. (See section 2.2.3.)

5) In a split body at prefix level k, the symbol "inner" represents those statements in the operation rule of X which belong to prefix levels inner to k, or a dummy statement if k = n. If none of C_1, ..., C_{n-1} has a split body the statements in operation rule of X are ordered according to ascending prefix levels.

A compound object could be described formally by a "concatenated" class declaration. The process of concatenation is considered to take place prior to program execution. In order to give a precise description of that process, we need the following definition.

An occurrence of an identifier which is part of a given block is said to be "uncommitted occurrence in that block", except if it is the attribute identifier of a remote identifier (see section 7.1), or is part of an inner block in which it is given a local significance. In this context a "block" may be a class declaration not including its prefix and class identifier, or a procedure declaration not including its procedure identifier. (Notice that an uncommitted identifier occurrence in a block may well have a local significance in that block.)

The class declarations of a given class hierarchy are processed in an order of ascending prefix levels. A class declaration with a non-empty prefix is replaced by a concatenated class declaration obtained by first modifying the given one in two steps.

1. If the prefix refers to a concatenated class declaration, in which identifier substitutions have been carried out, then the same substitutions are effected for uncommitted identifier occurrences within the main part.

2. If now identifiers of attributes defined within the main part have uncommitted occurrences within the prefix class, then all uncommitted occurrences within the main part of these identifiers are systematically changed to avoid name conflicts. Identifiers corresponding to virtual quantities defined in the prefix class are not changed.

The concatenated class declaration is defined in terms of the given declaration, modified as above, and the concatenated declaration of the prefix class.

1. Its formal parameter list consists of that of the prefix class followed by that of the main part.

2. Its value part, specification part, and virtual part are the unions (in an informal but obvious sense) of those of the prefix class and those of the main part. If the resulting virtual part contains more than one occurrence of some identifier, the virtual part of the given class declaration is illegal.

3. Its class body is obtained from that of the main part in the following way, assuming the body of the prefix class is a split body. The begin of the block head is replaced by a copy of the block head of the prefix body, a copy of the initial operations of the prefix body is inserted after the block head of the main part and the end of the

compound tail of the main part is replaced by a copy of the compound tail of the prefix body. If the prefix class body is not a split body, it is interpreted as if the symbols ";inner" were inserted in front of the end of its compound tail.

If in the resulting class body two matching declarations for a virtual quantity are given (see section 2.2.3), the one copied from the prefix class body is deleted.

The declaration of a label is its occurrence as the label of a statement.

Examples:

```
class point (x,y); real x,y;
   begin ref (point) procedure plus (P); ref (point) P;
                 plus :- new point (x+P.x, y+P.y);
   end point;
```

An object of the class point is a representation of a point in a cartesian plane. Its attributes are x,y and plus, where plus represents the operation of vector addition.

```
point class polar;
   begin real r,v;
         ref (polar) procedure plus (P); ref (point) P;
               plus :- new polar (x+P.x, y+P.y);
     r:= sqrt (x↑2+y↑2);
     v:= arctg (x,y);
end polar;
```

An object of the class polar is a "point" object with the additional attributes r,v and a redefined plus operation. The values of r and v are computed and assigned at the time of object generation. ("arctg" is a suitable non-local procedure.)

2.2.3 Virtual quantities

Virtual quantities serve a double purpose:

1) to give access at one prefix level of an object to attributes declared at inner prefix levels, and

2) to permit attribute redeclarations at one prefix level valid at outer prefix levels.

The following specifiers are accepted in a virtual part:

label, switch, procedure and <type> procedure.

A virtual quantity of an object is either "unmatched" or is identified with a "matching" attribute, which is an attribute whose identifier coincides with that of the virtual quantity, declared at the prefix level of the virtual quantity or at an inner one. The matching attribute must be of the same kind as the virtual quantity. At a given prefix level, the type of the matching quantity must coincide with or be subordinate to (see Section 3.2.5) that of the virtual specification and that of any matching quantity declared at any outer prefix level.

It is a consequence of the concatenation mechanism that a virtual quantity of a given object can have at most one matching attribute. If matching declarations have been given at more than one prefix level of the class hierarchy, then the one is valid which is given at the innermost prefix level outer or equal to that of the main part of the object. The match is valid at all prefix levels of the object equal or inner to that of the virtual specification.

Example:

The following class expresses a notion of "hashing", in which the "hash" algorithm itself is a "replaceable part". "error" is a suitable non-local procedure.

```
class hashing (n); integer n;
   virtual: integer procedure hash;
      begin integer procedure hash (T); value T; text T;
         begin integer i;
L:          if T.more then
            begin i := i+rank (T.getchar);
                  go to L;
            end;
            hash := i - (i ÷ n × n);
         end hash;
      text array table [0:n-1]; integer entries;
       integer procedure lookup (T,old); name old;
                         value T; Boolean old; text T;
          begin integer i;
                i:= hash(T);
L:              if table[i] == notext then
                begin table[i] :- T; entries :=
                                    entries+1; end
                else if table [i] = T then
                     old := true
                else if entries = n then
                        error("hash table
                              filled completely")
                else begin i := i+1;
                           if i = n then i := 0;
                           go to L
                     end;
                lookup := i;
          end lookup;
   end hashing;
```

```
hashing class ALGOL hash;
   begin integer procedure hash(T); value T;
                                    text T;
         begin integer i; character c;
         L: if T.more then   While T.more do
            begin c := T.getchar;
               if c ≠ '_' then
                     i := i + rank(c);
               go to L;
            end;
            hash := i-(i ÷ n × n);
         end hash;
   end ALGOL hash;
```