1. Repetitions (infix arrays).
++++++++++++++++++++++++++++++++

Repetitions are defined in the Simula program by means of the %INFIX-directive (meant to replace 'prior', also in procedure declarations). FEC will simply replace %INFIX with ' prior' (until we have a better idea!).

Example: %infix ref(symbolbox) array symboltable(0:127);

FEC
===

In order to simplify the implementation, such repetitions cannot occur as parameters (FEC will check and error-report). The generated S-code for an element access is as follows:

<static encloser of rep to stack>
REMOTE <repetition-tag>
<eval index>
INDEX (or INDEXV as the case may be)

- normal array element access (1-dim, lb=zero, no checking):

<static encloser of array to stack>
REMOTEV <array-tag>
REMOTE <arr1elttag(type)>
<eval index>
INDEX (or INDEXV as the case may be)

In addition, SCODER2 will generate the necessary structures for the GC (see below).

RTS
===

SIMOB has yet to be changed to be able to look at repetitions.

A new kind code for infix repetions is defined: K_REP (=6)
This is currently for SIMOB use only.

MAX_REP (=.....) defines the maximum number of elements in a repetition.

A pointer variable has been added to the prototype:

```
Visible record ptp;
begin ref(pntvec) refVec;
    ref(rptvec) repVec; -- NEW
    ref(ptpExt) xpp;
    size lng;
end;
```

The repetition vector is:

```
Visible record rptvec;
begin range(0:MAX_REP)   npnt;
    range(0:MAX_REP)   nrep;
    infix(repdes)      rep(0);
end;
```

- with infix descriptors:

```
Visible record repdes;   info  "TYPE";
begin range(0:MAX_REP)   nelt;
    range(0:MAX_TYPE)  type;
    field()            fld;
end;
```

In the repetition vector (rptvec), attribute 'npnt' counts the number of repetitions of interest to GC (in SIMULA, ref and text repetitions). 'nrep' is the total number of repetitions, i.e. npnt <= nrep. If SIMOB-level is less than 2, only ref-type repetitions are described (i.e. npnt=nrep allways).

'fld' of the repetition descriptor (repdes) designates its first element. Currently, a repetion must have lower bound zero.

Changes in STRG re repetitions. The following declarations are included in PASS1, PASS3 and the storage checker chckSTRG:

```
name(ref(inst)) qref; name(infix(txtqnt)) qtex;
ref(rptvec) repVec; range(0:MAX_REP) np;
infix(repdes) repdesc;
```

In PASS1, the following has been changed (the goto MARKref saves duplicated code at very little cost):

```
    when S_SUB,S_PRO,S_THK:
        pp:= x.pp; plv:=0; goto MARKref;

    when S_ATT,S_DET,S_RES,S_TRM,S_PRE:
        pp:= x.pp; plv:= pp qua ref(claptp).plv;
MARKref:
        MARK_and_FOLLOW(%x.sl%);
        MARK_and_FOLLOW(%x qua ref(inst).dl%);
        repeat refVec:=pp.refVec; if refVec <> none
            then j:=refVec.npnt;
                repeat while j <> 0
                do j:=j-1; q:= x + refVec.pnt(j);
         (*)        MARK_and_FOLLOW(%var(q qua name(ref()))%);
                endrepeat; endif;
```

```
        --- now scan and mark all ref-type repetitions
        if pp.repVec <> none
        then
            repVec:=pp.repVec;
            j:=repVec.npnt; repeat while j <> 0
            do j:=j-1;
              repdesc:=repvec.rep(j); np:=repdesc.nelt
              if repdesc.type=T_REF
              then
                  qref:=x+repdesc.fld;
                  repeat
(*)                       MARK_and_FOLLOW
(*)                       (%var(qref)%)
                  while np<>0 do np:=np-1;
                    qref:=name(var(qref)(1)) endrepeat
              else -- if repdesc.type=T_TXT
                  qtex:=x+repdesc.fld;
(*)                repeat MARK_AND_FOLLOW
(*)                       (%var(qtex).ent%);
                  while np<>0 do np:=np-1;
                    qtex:=name(var(qtex)(1)) endrepeat
              endif
            endrepeat
        endif
    while plv<>0 do plv:=plv-1;
              pp:= pp qua ref(claptp).prefix(plv);
    endrepeat;
```

PASS3 and chckSTRG are changed in the same manner - the lines marked (*) above are different, otherwise the code is an exact copy of the PASS1 code.

2. Literals (named Simula constants).

FEC

SCODER1E will generate descriptors of literals if SIMOB level > 2.

RTS

SIMOB has yet to be changed to be able to look at literals.

A new mode code for literals is defined: M_LIT (=5)
This is currently for SIMOB use only. (this code implies M_LOCAL)

Literals are described (to SIMOB) as a sub of atrdes. Since the only legal ref-const is NONE, the qualification is not needed. (but it may easily be added later, if necessary)

```
 Visible record litdes:atrdes; -- descriptor of named const
 begin infix(quant) qnt; end;  -- has fld=nofield and mode=M_LIT
```

'qnt' contains (or refers in case of text) the literal value.

3. Explicit alloc/dealloc of "foreign" objects.
++

These "objects" are intended for use outside the S-port part. e.g. as file buffers for ENVIR. The facility is not usable from Simula since the GADDR parameters cannot be handled there.

The RTS-procedures ALLOCO and FREEO will allocate and free these objects in the dynamic pool. The return value of ALLOCO is the GADDR of the first accessible byte. The object allocated is sub of 'inst', and carries the ALLOCO-parameter 'moveit' in the system head (see below). 'moveit' is a routine in the callers environment, which is informed (by GC) whenever GC intends to move the object. This routine may be different for each object ALLOCated, or it may be the same for all such objects. If not supplied, the system will error terminate during GC.

FEC
===

Unknown to FEC.

RTS
===

A new sort code for explicitly allocated "objects": S_ALLOC (=20)
The codes for S_SAV and S_GAP has been increased by one.

The following profiles and entry points are included:

```
--- Allocate - free - move  non-simula object  ---
Visible global profile Palloc; -- body in RTS'CENT
import integer nbytes; entry(Pmovit) movrut; export name() Nobj;
end;

Visible global profile Pfree; -- body in RTS'CENT
import name() Nobj;
end;

Visible global profile Pmovit; -- body in ENVIR
--- entry point is parameter to ALLOCO
import name() old, new;
end;

Visible entry(Palloc)         allocO    system "ALLOCO";
Visible entry(Pfree )         freeO     system "FREEOB";
```

The definition of the instance record has been changed as follows:

```
Visible record inst:entity;
begin variant ref(inst)        dl;
            label            lsc;
      variant entry(Pmovit)     moveIt;
end;
```

The routine entry is placed here so that the data space of the "non-object" is allocated at an OADDR - this could be done in different manners, but this method saves tags (and it does not

increase the allocated area for normal instances ?)

The explicitly allocated "non-objects" are of the following format,
sort code = S_ALLOC:

```
Visible record nonObj:inst;
begin character cha(0); end;
```

The bodies of ALLOC and FREE are included in RTS'CENT:

```
Visible body(Palloc) alloc;
-- import integer nbytes; entry(Pmovit) movrut;
-- export name() Nobj;
begin size lng; ref(nonObj) obj;
     lng:=rec_size(nonObj,nbytes); -- computed at runtime
     ALLOC2(S_ALLOC,obj,lng); obj.lng:=lng; obj.moveit:=movrut;
     Nobj:=name(obj.cha);
end;

Visible body(Pfree ) free ;
-- import name() Nobj;
begin ref(nonObj) obj;
     obj:=conv_ref(Nobj); obj.sort:=S_TEXT;
end;
```

The only change necessary in STRG (apart from adding S_ALLOC in case statements as if it was S_TXTENT) is in PASS4:

```
    when S_ALLOC:
        if x<>x.gcl
        then call Pmovit(x qua ref(inst).moveIt)
                    ( name(x qua ref(nonObj).cha) ,
                      name(x.gcl qua ref(nonObj).cha) ) endif
        x.gcl:=none; x:= x + x.lng
```