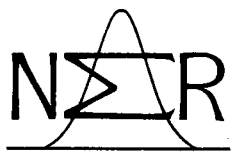
NORSK REGNESENTRAL
NORWEGIAN COMPUTING CENTER



Revised pages

SIMULA®

USERS GUIDE
IBM System/360

UNIVERSITETSFORLAGET
Oslo - Bergen - Tromsø

To make the use of subscripted variables very flexible, one is allowed to write arithmetic expressions in the subscript positions, so that if I = 2, for example, then LONGSIDE(2*I) and LONGSIDE(4) refer to the same quantity.

A CHARACTER ARRAY is used to represent the crossword. A blank is represented by the CHARACTER constant '␣' and a blocked out square by '*'. The declaration of the ARRAY is:

```
CHARACTER ARRAY CWORD (0:N+1, 0:N+1)
```

representing the crossword (dimensions 1 through N), and the rim (dimensions 0 and N+1). This will make available $(N+2)^2$ variables of type CHARACTER each initialised to the initial value '^', binary zero (CHAR(0)). N.B. several CHARACTERS are not visible when printed - this is one of them. It is not the same CHARACTER as blank. To distinguish, blank is written '␣'.

The strategy is (in words):

1) N, the number of rows and columns of the crossword itself is given on the first card

2) Read in the data from cards - each row being represented on a fresh card

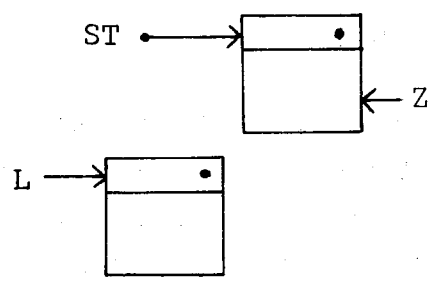
   Characters may be read in one by one by successive calls on INCHAR.

The coding is:

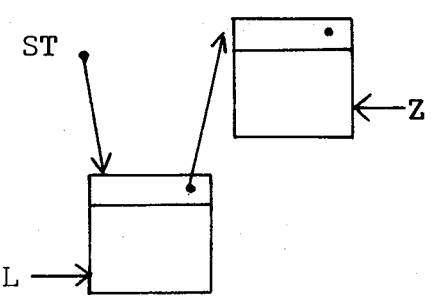
```
L.NEXT :- ST;
ST :- L;
COMMENT ***NOW L IS INSERTED*** ;
C.NEXT :- ST;
ST :- C;
COMMENT ***NOW C IS INSERTED*** ;
```

Snapshots before and after the insertion of L are:

before



after



The coding is a repeat except for a parameter saying which object is to be inserted. The PROCEDURE concept enables us to give CLASS STACKABLE procedure attribute "INTO", and then we can write

```
L.INTO (ST)
C.INTO (ST)
```

for the same effect. This is not only more concise, but more readable than before. This time we have to be rather careful

f) separators

name	graphic	use
comma	,	separating elements in lists
dot	.	denoting decimal point in REAL numbers; remote accessing
colon	:	follows labels; follows VIRTUAL; separates array bounds in array declarations
becomes	:=	in value assignments
denotes	:-	in reference assignments
semicolon	;	separates declarations and statements; separates various parts of procedure and class headings
dollar	$	may be used instead of a semicolon
blank	␣	used as a separator
hash	#	precedes a hexadecimal constant
underscore	_	used in identifiers (e.g. RATE_OF_PAY)

2.4 THE USE OF BLANKS

Identifiers, arithmetic constants, composite operators (e.g. =/=), key words (e.g. BEGIN) may not contain blanks. Blanks are permitted as CHARACTER-constants and in TEXT-constants.

Identifiers, constants and key words may not be immediately adjacent. They must be separated by an arithmetic operator, parenthesis ("(" or ")"), reference comparator, negation (-), non-key-word relational operator (<,<=,=,¬=,>,>=,==,=/=), comma, dot, colon, becomes symbol (:=), denotes symbol (:-), semicolon, or blank. Moreover additional intervening blanks are always permitted.

Examples:

X + Y	is equivalent to X+Y
A (I)	is equivalent to A(I)
A :=X :=Y	is equivalent to A:=X:=Y

b) IF X > 0 THEN BEGIN .....
END OF TRUE PART
ELSE BEGIN .....
END OF FALSE PART;

Where the strings "OF TRUE PART" and "OF FALSE PART" are treated as comments.

c) X := X COMMENT**THAT WAS X;**2 COMMENT**SQUARED;;
is equivalent, as regards program execution, to
X := X**2;

3 IDENTIFIERS

An identifier is a string of alphanumeric or underscore characters, not contained in a comment or constant, preceded and followed by a delimiter - the initial letter must always be alphabetic.

identifier

letter [letter|digit|_]...

Examples:

valid identifiers

X
SIMULA_67
A15
MORGAN_PLUS_4
APPLE-

invalid identifiers

END	reserved for use as a keyword
SYM␣BOL	contains a blank
3C	does not begin with a letter

(SHORT)INTEGER-constant

decimal-digits

The range of values is the set of whole numbers from 0 through $2^{31}-1$ (= 2147483447). If the magnitude lies in the range 0 through $2^{15}-1$ (= 32767), the constant is treated as a SHORT INTEGER constant, if the magnitude lies in the range 2^{15} through $2^{31}-1$, it is treated as an INTEGER constant. If the magnitude is equal to or exceeds 2^{32}, the number is interpreted as a REAL constant.

Examples:

```
     0      SHORT INTEGER
    91      SHORT INTEGER
814728      INTEGER
```

TEXT-constants have the form

"{any sequence of members of the data character set}"

CHARACTER-constants are represented by

'{any one member of the data character set}'

Examples:

valid CHARACTER-constants:

'X'	
'&'	
'␣'	
'''	the character quote itself

invalid CHARACTER-constants:

':-'	two data character set members
'A␣'	blanks are significant in character constants

If one wishes to represent a text quote inside a TEXT-constant, it is represented by "" - two adjacent text quotes. Notice that """" has length 1, """""" has length 2 etc.

Each variable declared in a type declaration has an initial value (given in the table below). Thereafter the value of a variable is the one last assigned to it, or if no assignment has yet been made, the initial value.

Type	Initial value	Assignable range
INTEGER	0	Whole number in the range -2^{31} through $2^{31}-1$.
SHORT INTEGER	0	Whole number in the range -2^{15} through $2^{15}-1$.
REAL	0.0	Number in the range $\pm(0 \rightarrow 10^{75}$ approx.) to ∿7 decimal places.
LONG REAL	0.0	Number in the range $\pm(0 \rightarrow 10^{75}$ approx.) to ∿16 decimal places.
BOOLEAN	FALSE	TRUE, FALSE.
CHARACTER	CHAR(0)	CHAR(0), CHAR(1), ... CHAR(255).
REF(CLASS-identifier)	NONE	NONE or any object of the qualifying class or included in the qualifying class.
TEXT	NOTEXT	NOTEXT or any string of characters from the data character set of length 0 through (2^{15}-20) characters.

function-declaration

```
REF(POINT) PROCEDURE ADD(Q); REF(POINT)Q;
   IF Q =/= NONE THEN ADD :- NEW POINT(X+Q.X,Y+Q.Y)

REAL PROCEDURE NORM(A,N); REAL ARRAY A; INTEGER N;
BEGIN  REAL T; INTEGER I;
       FOR I := 1 STEP 1 UNTIL N DO
           T := T + A(I)**2;
       NORM := SQRT(T)
END ***NORM***

INTEGER PROCEDURE FACTORIAL(N); INTEGER N;
IF N < 0 THEN ERROR ELSE
   IF N < 2 THEN FACTORIAL := 1
            ELSE FACTORIAL := N*FACTORIAL(N-1)
```

A function returns a value of the type indicated in its declaration, and may be used wherever a value of that type is legal. (It may also be used as a statement in which case the function value is ignored);

```
P :- R.ADD(S)

X := NORM(MATRIX, 10)

IF NORM (MATRIX, 10) < &-6 THEN
   OUTTEXT ("ELEMENTS␣ALL␣ZERO")
```

Call by name

Call by name is an optional transmission mode available for parameters to procedures, but not to classes. It represents a textual replacement in that the formal-parameter may be considered replaced throughout the PROCEDURE-body by the corresponding actual-parameter.

Whereas call by value and call by reference operate on variables local to the PROCEDURE-body itself, call by name operates on non-local quantities and can alter global quantities. It is therefore especially useful for the controlled alteration of several variables external to the PROCEDURE-body.

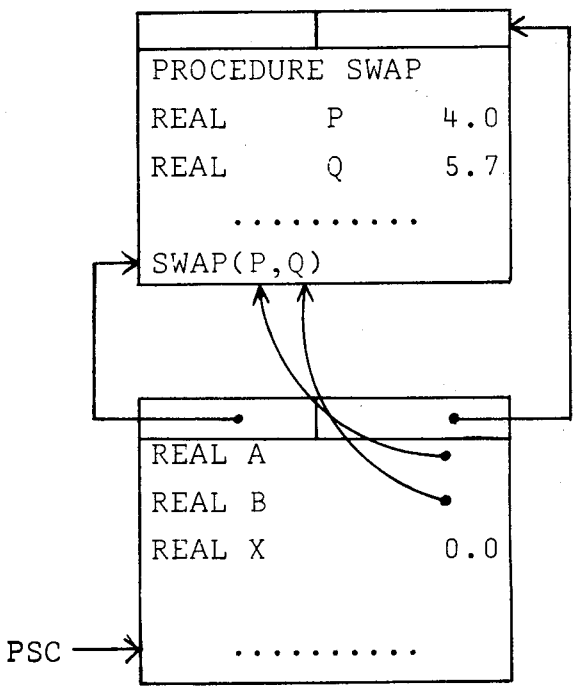
The following rules apply:

1) the type of a name parameter is that prescribed by the corresponding formal specification.

2) if the type of the actual-parameter does not coincide with that of the formal specification, then an evaluation of the expression is followed by an assignment of the value or reference obtained to a fictitious variable of the latter type. This assignment is subject to the rules of section 2.7.2. The value or reference obtained by the evaluation is the contents of the fictitious variable.

Section 2.7.2 defines the meaning of an assignment to a variable which is a formal-parameter called by name, or is a subscripted variable whose array identifier is a formal-parameter called by name, if the type of the actual parameter does not coincide with that of the formal specification.

A snapshot at the procedure call is:



Program Sequence Control (PSC) references the current statement

No local copies are made. Every occurrence of A or B in the PROCEDURE-body means a re-evaluation of the actual-parameter. Notice that the actual-parameters are evaluated in the context of the procedure call.

The result of executing the call on SWAP is to set the value of P to 5.7 and the value of Q to 4.0.

The available transmission modes for legal parameters to classes are shown in the following table:

CLASS PARAMETERS

Parameter	Transmission	
	call by value	call by reference
value-type	D	X
reference-type	X	D
TEXT	O	D
value-type ARRAY	O	D
reference-type ARRAY	X	D

D : default mode
O : optional
X : illegal

The transmission modes "call by value" and "call by reference" are explained in section 2.5.4.

The discussion of CLASS-declarations begins by considering two selected examples of increasing scope.

The expression

NEW A(...)

creates an object of the class A (the order and number of the actual-parameters must correspond with the order and number of the formal-parameters) and commences execution of its actions. The execution continues until the final END of the class body is encountered, when the execution is terminated. However this may be interrupted in four ways - by a GOTO-statement (which leads out of the object), or by calls on the system procedures "DETACH", "RESUME" or "CALL". A GOTO exit will leave the object in the terminated state. "Detach" suspends the actions of the class body and names it an independent component of the program. Its actions may be continued later by a call on "resume". The object is "attached" when entered on generation by execution of NEW or when it is the subject of a call on CALL.

5.6 EXTERNAL DECLARATIONS

External declarations allow the inclusion in a program of previously compiled procedures and classes. By judicious use of this feature a programmer can build up a library of building blocks on which to base future programs.

External FORTRAN and Assembly procedure declarations may also be important for the following reasons:

1) efficiency. Minimal coding may be achieved in critical regions of a program.

2) features of a machine which are hidden by SIMULA may be accessed.

3) routines (already written) in other languages may be incorporated into SIMULA programs.

external-declaration

EXTERNAL { [FORTRAN] [ASSEMBLY] [type] PROCEDURE } id-list
{ CLASS }

Examples

```
EXTERNAL ASSEMBLY INTEGER PROCEDURE BITS_ON
EXTERNAL REAL PROCEDURE MIN, MAX
EXTERNAL PROCEDURE READ, WRITE
EXTERNAL FORTRAN PROCEDURE FORMAT
EXTERNAL CLASS DRAUGHTING
```

Preparation of external procedures

In order to prepare external procedures written in SIMULA, they should be compiled one at a time by compiling with PARM = EXTERN or PARM = 'EXTERN = P'. Note that the syntax of a separately compiled SIMULA procedure is identical with that of a procedure declaration, i.e. the program does not start with BEGIN! Separately compiled procedures are placed in a user specified library (see "SIMULA Programmer's Guide IBM System 360", section 2.4, p. 30 for examples).

An external procedure may be saved as an object module or as a load module.

Note the following restrictions:

1) FORTRAN, ASSEMBLY procedures

Parameter checking is the responsibility of the writer of the external procedure since it cannot be wholly done by the compiler. In FORTRAN procedures, only FORTRAN parameters may be transmitted.

In ASSEMBLY procedures, only literals, simple or array variables are allowed as actual parameters. For details concerning parameter retrieval and other aspects relevant to the writing of FORTRAN and ASSEMBLY procedures, consult Programmer's Guide, Appendix G.

2) External procedures may not contain local classes and object reference-variables.

Preparation of external classes

In order to prepare external classes, they should be compiled one at a time by compiling with PARM = 'EXTERN = C'. Note that the syntax of a separately compiled class is identical to that of a class declaration. Separately compiled classes are placed in a user specified source module library with DD-name SYSPUNCH.

Note the following restriction:

A separately compiled class must not contain either COPY code, or other external class declarations.

Use of externally compiled procedures and classes

Retrieving an externally compiled class is achieved by specifying the library containing it as a data set with DDNAME SYSLIB to the SIMULA compiler.

The retrieval of an externally compiled procedure is achieved by specifying the library containing it as a data set concatenated with SYSLIB of the loader or linkage editor.

Let X be a simple object expression qualified by class C, and A an attribute identifier.

The X.A is valid if X =/= NONE and class C has at least one attribute identified by A. If there is only one attribute with identifier A contained in class C or in its prefix chain, then it is designated by X.A. If class C contains more than one attribute with identifier A, then X.A designates the attribute with identifier A at the innermost prefix level.

N.B. The main part of any class declaration can contain at most one attribute with that identifier.

Example:

```
    REF(C1)X1; REF(C2)X2; REF(C3)X3; REF(C4)X4

    CLASS C1(A); REAL A;
    BEGIN ..................... END;
C1  CLASS C2;
    BEGIN  TEXT A; ... END;
C2  CLASS C3;
    BEGIN  ..................... END;
C3  CLASS C4;
    BEGIN  BOOLEAN A; ........ END;
X1 :- X2 :- X3 :- X4 :- NEW C4(6.0);
```

6.2 FUNCTION DESIGNATORS

A function designator denotes the value obtained by evaluating the associated procedure body when supplied with the associated actual parameter list (if any).

function-designator

```
{identifier        }
{remote-identifier } [actual-parameter-list]
```

actual-parameter-list

```
(actual-parameter[, actual-parameter]...)
```

actual-parameter

```
{expression           }
{ARRAY-identifier     }
{SWITCH-identifier    }
{PROCEDURE-identifier }
{LABEL-identifier     }
```

Examples:

```
ININT
INTEXT(20)
SIN(A+B)
H.FIRST
CURRENT.NEXTEV.SUC
POINT(5).MODULUS
```

Two text values are equal if they are both empty or if they are both instances of the same character sequence. Otherwise they are unequal, and then a text value T ranks lower than a text value U if one of the following conditions is fulfilled.

1) T is empty
2) U is equal to T followed by one or more characters
3) If the first i-1 characters of T and U are the same, and the ith character of T ranks lower than the ith character of U.

Examples:

```
NOTEXT = ""          TRUE
"0"  <  "9"          TRUE
"ABCDE"="ABCDEF"     FALSE
"+12" =/= "=12"      TRUE
"ABC"  =  "ABC␣"     FALSE
```

When a block is prefixed, the identifiers declared in the corresponding CLASS are made available. Nevertheless, an identifier in the CLASS may be redefined in the main-block or compound-statement.

The execution of a block is as follows:

step 1: if the block is prefixed then the actual parameters if any are evaluated.

2: if the declarations of the block contain array bounds then these are evaluated. (They may make reference to parameters of the prefix).

3: Execution of the statements of the block body begins with the first statement of the prefix, if any, otherwise with the first statement of the main block. After execution of the block body (unless it is a GOTO statement) a block exit occurs and the statement textually following the entire block is executed.

A CLASS identifier possibly followed by an actual parameter list can prefix a main-block or compound-statement. This results in concatenating the object of the stated class with that main-block or compound-statement, which means that the capabilities of the stated class and its including classes are available within that main-block or compound statement.

When an instance of a prefixed block is generated, the formal parameters of the class are initialised as indicated by the actual parameters of the block prefix. A virtual quantity is identified by the quantity defined by a matching declaration in the block head of the main-block or compound-statement, or by the matching definition at the innermost prefix level of the prefix sequence. The operation rule of the concatenated object is defined by principles similar to those given in section 5.5.

reference-assignment

```
{variable :-                 }        object-expression
{PROCEDURE-identifier :-     } ...    TEXT-expression
```

Examples:

```
P.X[1] :- NONE
P :- Q :- THIS POINT.SUC
ADD :- NEW POINT
THIS LINE.P :- NONE
T :- BLANKS(80)
S :- R :- T.SUB(1,20)
```

Consider the object-reference-assignment

```
V :- object-expression
```

Both the left part and the right part have a qualification. Let these be Q_ℓ and Q_r respectively - note that NONE is here considered as having a universal qualification which is inner to every other qualification. The situations that can arise are illustrated in the context:

```
REF(CHAIN)C;      CLASS CHAIN.....;
REF(MEMBER)M;     CLASS MEMBER.....;
REF(POINT)P;      MEMBER CLASS POINT....;
REF(LINE)L;       MEMBER CLASS LINE.....;
```

Case 1: Q_ℓ is equal or outer to Q_r

e.g.

```
A)    M :- NEW MEMBER;
B)    M :- NEW POINT;
C)    M :- P;
D)    M :- NONE;
```

The assignment is legal. In B), the attributes of the referenced POINT-object may be accessed by use of QUA as in

```
M QUA POINT.X
```

or by use of INSPECT as in either of

```
INSPECT M QUA POINT DO ... X ...
INSPECT M WHEN POINT DO ... X ...
```

Case 2: Q_ℓ is inner to Q_r

e.g.

```
A)      P :- NEW MEMBER;
B)      P :- M;
```

The assignment may be legal. A) is clearly not, but B) is legal if M is currently referencing a POINT object or an object of a class inner to POINT, e.g. after

```
M :- NEW POINT...;
```

This must be checked at runtime.

In cases 1 and 2, the qualifications Q_ℓ and Q_r are said to be compatible. Notice that compatibility is decided at compile time.

Case 3: Q_ℓ and Q_r are not compatible.

e.g.

```
A)      P :- NEW LINE.....;
B)      P :- NEW CHAIN;
C)      M :- C;
```

The assignment is illegal.

Similar rules apply to the object-reference assignments implicit in the passing of parameters (FP :- AP) and assignments to the result variable in a function procedure body.

Multiple assignments take the form

v_1 :- v_2 :- .... :- v_n :- object-expression

and are equivalent to

v_n :- object expression,

v_{n-1} :- v_n;

.....

v_1 :- v_2;

The considerations above apply at each step.

The fact that an object-reference-assignment is always checked for legality (mainly at compile time) has the following implication. For any object-reference-variable or indeed, object-expression whose value is V and with qualification Q_v, the following is always true under program execution:

(V == NONE) OR (V IN Q_v)

Example:

The program below prints out the first three verses of "The Twelve Days of Christmas". The program logic is built around a SWITCH.

```
BEGIN  SWITCH CASE := LINE1, LINE2, LINE3;
       INTEGER VERSE;
       TEXT ARRAY T(1:3);
       T(1) :- COPY("FIRST");
       T(2) :- COPY("SECOND");
       T(3) :- COPY("THIRD");

IF_WE_USE_FOR_HERE_THE_LABELS_BECOME_INVISIBLE:
       VERSE := VERSE + 1;
       OUTTEXT("ON␣THE␣"); OUTTEXT(T(VERSE));
       OUTTEXT("␣DAY␣OF␣CHRISTMAS"); OUTIMAGE;
       OUTTEXT("MY␣TRUE␣LOVE␣SENT␣TO␣ME"); OUTIMAGE;
       GOTO CASE(VERSE);
LINE3: OUTTEXT("THREE␣FRENCH␣HENS"); OUTIMAGE;
LINE2: OUTTEXT("TWO␣TURTLE␣DOVES,␣AND"); OUTIMAGE;
LINE1: OUTTEXT("A␣PARTRIDGE␣IN␣A␣PEAR␣TREE"); OUTIMAGE;
       EJECT(LINE + 2);
       IF VERSE < 3 THEN
          GOTO IF_WE_USE_FOR_HERE_THE_LABELS_BECOME_INVISIBLE;
END
```

DUMMY STATEMENTS

A dummy-statement executes no actions. Its main use is to place a label before an END

dummy-statement

Examples:

```
IF X > 0 THEN ELSE X := -X;

BEGIN ..... LAB : END;
```

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 GOTO β;
    S;
    σ := A2;
    C := C + σ;
    GOTO α;
β : next for list element
```

3. V WHILE B

```
α : C := V;
    IF ¬ B THEN GOTO β;
    S;
    GOTO α;
β : next for list element
```

4. 0

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

5. 0 WHILE B

```
α : C :- 0;
    IF ¬ B THEN GOTO β;
    S;
    GOTO α;
β : next for list element
```

PROCEDURE STATEMENTS

A procedure-statement calls for the execution of the PROCEDURE-body. Before execution, the formal-parameters of the procedure are replaced by the actual-parameters.

PROCEDURE-statement

[simple-object-expression.
simple-TEXT-expression.] PROCEDURE-identifier [(expression
[, expression]...)]

Examples:

```
INTO(H)
OUTTEXT("***0***")
SYSIN.INIMAGE
```

The procedure statement must have the same number of actual parameters in the same order as the formal-parameters of the procedure heading.

Restrictions

1) An actual-parameter corresponding to a formal-parameter called by NAME which is assigned to within the PROCEDURE-body must be a variable.

2) If the formal-parameter is an ARRAY (PROCEDURE), then the number of dimensions (actual-parameters) used within the PROCEDURE-body must correspond to the number of dimensions (actual-parameters) of the actual ARRAY (PROCEDURE).

Connection statement

```
                                        ⎧      statement                          ⎫
INSPECT object-reference DO             ⎨[WHEN CLASS-identifier DO statement]...⎬
                                        ⎩                [OTHERWISE statement]  ⎭
```

Examples:

```
INSPECT SYSOUT DO
BEGIN  OUTTEXT("TITLE");
       OUTIMAGE;
       EJECT(LINE+10);
END;
```

```
INSPECT X WHEN A DO OUTTEXT("X IN A")
          WHEN B DO OUTTEXT("X IN B")
          OTHERWISE OUTTEXT("ERROR");
```

```
INSPECT X DO
  INSPECT Y DO P := Q
            OTHERWISE OUTTEXT("Y==NONE");
```

To avoid ambiguity, an OTHERWISE refers back to the nearest INSPECT.

The remote accessing of objects of classes may be accomplished by the dot notation or by connection. In most cases the methods are interchangeable, but if the object contains a class attribute at a certain level, then attributes at that level and levels inner to it can only be accessed by connection.

In addition to sequencing PROCEDURES, "DETACH" and "RESUME", there is the PROCEDURE CALL which has one reference parameter which must be a reference to a detached object.

The execution of CALL(Y) from within a block X, will "attach" the detached object Y to X and continue execution of the actions of Y.

The detailed description of program sequencing given in the "SIMULA 67 Common Base Language" is not repeated here. Further enquires are directed to that document §9.

```
CLASS HAND;
BEGIN  PROCEDURE PLACE(C); REF(CARD)C;
       BEGIN  REF(HEAD)S; REF(CARD)X;
              S :- SUIT(C.COLOUR);
              IF ¬ S.EMPTY THEN
              BEGIN  X :- S.FIRST QUA CARD;
                     WHILE X =/= NONE DO
                     BEGIN  IF X.RANK > C.RANK THEN
                                BEGIN  C.PRECEDE(X);
                                       GOTO L;
                                END;
                            X :- X.SUC;
                     END;
              END;
              COMMENT ***WE ENTER HERE IF S IS EMPTY OR
                         IF C.RANK IS THE HIGHEST MET
                         SO FAR*** ;
              C.INTO(S);
   L:  END ***PLACE*** ;

       REF(HEAD) ARRAY SUIT(1:4);
       SUIT(1) :- NEW HEAD;
       SUIT(2) :- NEW HEAD;
       SUIT(3) :- NEW HEAD;
       SUIT(4) :- NEW HEAD;
END ***HAND*** ;
```

actions are executed. When the active phase is over, that PROCESS may be rescheduled for a later active phase (for example, by REACTIVATE or HOLD) or removed from the timing tree (by PASSIVATE or WAIT). It is apparent that RESUME is too primitive for this purpose as it involves rescheduling or removing EVENT NOTICES as well as switching the PSC from one PROCESS object to another.

However RESUME and DETACH do form the basis for the scheduling procedures. To prevent the user from destroying system security, event notices may not be explicitly referenced by the user - he must use the system procedures for scheduling or rescheduling. In addition, it is strongly recommended that explicit use of "DETACH", "RESUME" and "CALL" be avoided within a SIMULATION block.

There is one special PROCESS object which plays a key role in any SIMULATION - one referenced by MAIN. Whenever MAIN becomes CURRENT, it causes the actions of the SIMULATION block itself to be continued. The corresponding event notice can then be rescheduled (typically by a call on HOLD) and then the action switches from the SIMULATION block to the new CURRENT. Thus the SIMULATION block is itself treated as a program component during the SIMULATION.

PROCEDURE CANCEL(X); REF(PROCESS)X;.....;
CANCEL(P) where P is a reference to a PROCESS object will delete the corresponding event notice if any. If P is currently active or suspended, it thus becomes passive. If P is a reference to a passive or terminated PROCESS object or NONE, CANCEL(P) has no effect. Thus CANCEL(CURRENT) is equivalent to PASSIVATE.

PROCEDURE ACTIVATE

For user convenience, calls on the procedure ACTIVATE are written in terms of the corresponding activation-statements.

activation-statement

```
{ ACTIVATE   }
{ REACTIVATE } PROCESS-expression1 [{AT|DELAY} time [PRIOR]                    ]
                                   [{BEFORE|AFTER} PROCESS-expression2         ]
```

Let X be the value of PROCESS-expression1. If the activator ACTIVATE is used, then the activation-statement will have no effect (other than evaluating X) unless X is passive. If the activator REACTIVATE is used, then X may be active, suspended, or passive (in which latter case, the activation-statement acts as an ACTIVATE statement).

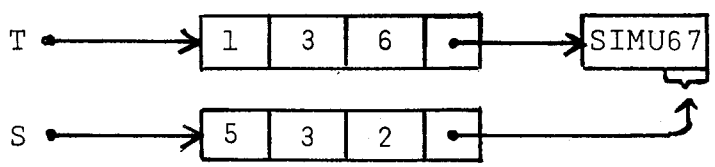
The type of scheduling is determined by the scheduling clause.

A complete description of the program now follows:

```
BEGIN  INTEGER POPULATION, LENGTH, CONTACTS, INCUBATION,
               U1, U2, U3, U4, UNINFECTED;
       REAL PRINF, PROBMASS, SIMPERIOD;
       COMMENT ***THE RANDOM STREAM NUMBERS ARE READ IN*** ;
       U1 := ININT; U2 := ININT;
       U3 := ININT; U4 := ININT;
       UNINFECTED := POPULATION := ININT;
       INCUBATION := ININT; LENGTH := ININT;
       CONTACTS := ININT;
       SIMPERIOD := INREAL;
       PRINF := INREAL; PROBMASS := INREAL;
SIMULATION BEGIN  REAL ARRAY PROBTREAT(1:LENGTH);
                  PROCESS CLASS SICKP;
                  BEGIN  INTEGER DAY;
                         BOOLEAN SYMPTOMS;
                         REF(HEAD)ENV;
                         PROCEDURE INFECT(N); INTEGER N;
                         BEGIN  INTEGER I;
                                FOR J := 1 STEP 1 UNTIL N DO
                                    IF DRAW(PRINF*UNINFECTED/
                                            POPULATION,U3) THEN
                                    BEGIN  NEW SICKP.INTO(ENV);
                                           ACTIVATE ENV.LAST;
                                    END;
                         END ***INFECT*** ;
```

After S.PUTCHAR('6');
S.PUTCHAR('7')

the snapshot is



Note that the value of T has been changed. The CP of S is now out of range. A further call

S.PUTCHAR
or S.GETCHAR

will result in a run time error. To provide a check, a BOOLEAN PROCEDURE MORE is provided which returns FALSE if the CP is out of range and TRUE otherwise. Currently,

T.MORE = TRUE S.MORE = FALSE

Other useful system defined procedures are:

LENGTH which returns the length of the currently referenced value
(T.LENGTH = 6
S.LENGTH = 2)

POS which returns the value of the CP
(T.POS = 3
S.POS = 3)

CHARACTER PROCEDURE GETCHAR;

The value of X.GETCHAR is a copy of the currently accessible CHARACTER of X provided X.MORE is TRUE. In addition, the position indicator of X is then increased by one. A run time error results if X.MORE is FALSE.

PROCEDURE PUTCHAR(C); CHARACTER C;

The effect of X.PUTCHAR(C) is to replace the currently accessible CHARACTER of X by the value of C provided that X.MORE is TRUE. In addition the position indicator of X is then increased by one. If X.MORE is FALSE, a run time error results.

Example:

The PROCEDURE COMPRESS rearranges the CHARACTERS of the TEXT object referenced by the actual parameter by collecting non-blank CHARACTERS in the leftmost part of the TEXT object and filling in the remainder, if any, with blanks. Since the parameter is called by reference (and not by name), its position indicator is unaltered.

```
PROCEDURE COMPRESS(T); TEXT T;
BEGIN  TEXT U; CHARACTER C;
       T.SETPOS(1); U :- T;
MOVELEFT:  WHILE U.MORE DO
           BEGIN  C := U.GETCHAR;
                  IF C ¬= '␣' THEN T.PUTCHAR(C);
           END;
           COMMENT ***WE NOW FILL IN THE RIGHT WITH
                      BLANKS*** ;
           T.SUB(T.POS,T.LENGTH-T.POS+1) := NOTEXT;
END ***COMPRESS***
```

b) TEXT-value-assignment

Consider the value assignment

```
T := P;
```

let the length of T be L1, and the length of the right part be a TEXT value of length Lr. There are three cases to consider:

L1 = Lr:	the character contents of the right part TEXT are copied to the left part TEXT
L1 > Lr:	the character contents of the left part are copied into the leftmost Lr characters of the left part TEXT, whose remaining L1-Lr CHARACTERS are filled with blanks.
L1 < Lr:	a run time error results.

After `T :- COPY("EIGHT␣CHARS");`
`T := "WRONG:11";`

then `T = "WRONG:11␣␣␣"`

Note that

```
T := NOTEXT;
```

would set all the character positions of T to blanks.

In a multiple TEXT value assignment

```
T1 := T2 := ..... TN := P;
```

then

T_J.LENGTH >= T_{J+1}.LENGTH

for J = 1,2,...,N-1

De-editing procedures

A de-editing procedure operating on a given TEXT reference X operates in the following way:

1) the longest numeric item of the given form is located, contained within X and containing the first character of X. If such an item can be found, a run time error results.

2) the numeric item is interpreted as a number. If it is outside the accepted range (see PART 2, section 5.1 a run time error results.

3) the position indicator of X is made one greater than the last character of the numeric item.

N.B. Unless otherwise stated, the de-editing procedures are illustrated in the context:

```
T :- COPY("1234.5+7.3&4AB");
S :- T.SUB(7,6);
R :- T.SUB(5,2);
```

INTEGER PROCEDURE GETINT;

Locates an integer-item.

```
T.GETINT = 1234
S.GETINT = 7
R.GETINT   causes a run time error
```

Editing procedures

Editing procedures in a given text reference X convert arithmetic values to numeric items. After an editing operation, the numeric item obtained is right adjusted in the TEXT X preceded by padding blanks. The final value of the position indicator is X.LENGTH+1.

A positive number is edited with no sign. If X == NOTEXT then a run time error results, otherwise if X is too short to contain the numeric item, an edit overflow is caused (X is filled with asterisks) and a warning message is given at the end of program execution.

```
Let     T = BLANKS(10);
```

PROCEDURE PUTINT(I); INTEGER I;

T.PUTINT(VAL) converts the value of the parameter to an integer-item of the designated value.

```
T.PUTINT(-37)          ␣␣␣␣␣␣␣-37
T.PUTINT(118.8)        ␣␣␣␣␣␣␣119
```

PROCEDURE PUTFIX(R,N); REAL R; INTEGER N;

T.PUTFIX(VAL,M) results in an integer-item of M=0, or a real-item (with no exponent) if M>1 with M digits after the decimal point. It designates a number equal in value to VAL rounded to M decimal places. A run time error results if M<0.

```
T.PUTFIX(18,0)         ␣␣␣␣␣␣␣␣18
T.PUTFIX(-1375.4,3)    ␣-1375.400
```

The PROCEDURES OPEN and CLOSE, which are specified as VIRTUAL but have no matching declaration at the "file" level, complete the definition of CLASS file. The matching PROCEDURES declared in the subclasses of "file" conform to the patterns below with possible minor variations depending upon the subclass. The variations are listed in the appropriate following sub-sections.

The PROCEDURE outlines are:

```
PROCEDURE OPEN(BUF); TEXT BUF;
BEGIN  IF "open" THEN ERROR;
       IMAGE :- BUF;
END
PROCEDURE CLOSE;
BEGIN ..........
       IMAGE :- NOTEXT;
END
```

No information can be processed through a "file" object until it has not only been generated but also opened. This can only be achieved by a call on the PROCEDURE OPEN whose actual parameter is assigned to IMAGE and acts as the buffer. A call on OPEN when a "file" is already open gives a run time error.

The PROCEDURE CLOSE closes a file and releases the buffer (by the assignment IMAGE :- NOTEXT). No information may be transmitted through a closed "file" object, but it may be opened again by a further call on OPEN.

than IMAGE, it is left adjusted and the remainder of IMAGE is filled with blanks. Finally the position indicator of IMAGE is set to 1. When the last record has been read in, and INIMAGE is called again, a call on ENDFILE will return TRUE. Any further call on INIMAGE, INCHAR, INTEXT, ININT, INREAL or INFRAC will result in a run time error.

BOOLEAN PROCEDURE LASTITEM;

returns FALSE only if the external file contains more information (non-blank CHARACTERS). It scans past all blank CHARACTERS (calling INIMAGE if need be). If LASTITEM returns FALSE then the currently accessible CHARACTER of IMAGE is the first non-blank CHARACTER. If ENDFILE returns TRUE, a call on LASTITEM also returns TRUE.

CHARACTER PROCEDURE INCHAR;

gives access to the next available CHARACTER and scans past it. If IMAGE.MORE is FALSE, then INIMAGE is called once and the value of the call is the first CHARACTER of the new image. INCHAR gives a run time error if an attempt is made to read past the last record in the file.

TEXT PROCEDURE INTEXT(W); INTEGER W;

INTEXT(M) creates a copy of the next M CHARACTERS (which may be spread over several records) and returns a reference to this copy. If M < 0 or M > 2^{15}-20 then a run time error results. A run time error will also result if the file does not contain M more CHARACTERS, i.e. an attempt is made to read past the last record.

The remaining PROCEDURES treat the file as a continuous stream of records. They scan past any number of blanks (calling INIMAGE if need be) and then de-edit a numeric item lying in one image. This is done by calling LASTITEM (which scans past the blanks)

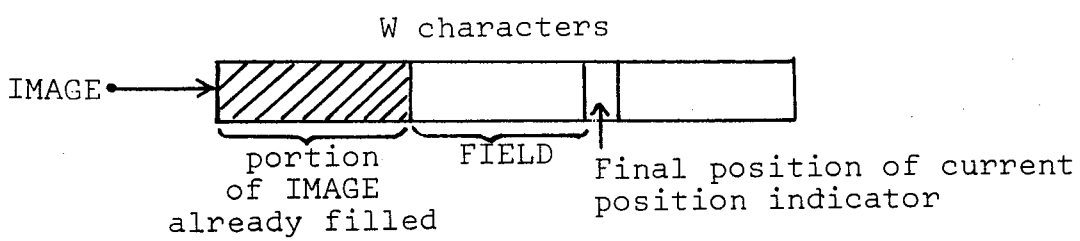
PROCEDURE OUTTEXT(T); VALUE T; TEXT T;

A copy of the CHARACTER sequence represented by the actual parameter is edited into IMAGE from the current position. If the remaining length of IMAGE is insufficient, OUTIMAGE is called and the editing process proceeds. Thus the TEXT value may be split over several external records.

PROCEDURE OUTCHAR(C); CHARACTER C;

Outputs the value of C into the current position of IMAGE (if MORE = FALSE, then OUTIMAGE is called first). In either case, the current position indicator is then incremented.

The remaining PROCEDURES are all based upon the PUT-PROCEDURES local to TEXTs. The corresponding OUT-PROCEDURES are augmented by an extra parameter W which specifies the field within.



The editing PROCEDURE commences by establishing a temporary TEXT reference (FIELD) to the next sequence of W CHARACTERS lying in one IMAGE. If the current IMAGE has not enough space left, OUTIMAGE is called. Then the value is edited by calling FIELD."PUT***" where "PUT***" is the PUT-PROCEDURE corresponding to the OUT-PROCEDURE. Finally the current position indicator is increased by W to reference past FIELD - past the just-edited field.

PROCEDURE EJECT(N); INTEGER N;

This PROCEDURE skips to a certain line on the page - (it avoids calling OUTIMAGE several times). EJECT(L) will position to line L on this page if this is further down the current page (if L > LINE), or else skip to LINE L of the next page if L <= LINE.

A run time error occurs if L <=0. If L > LINESPERPAGE, EJECT(L) is equivalent to EJECT(1).

INTEGER PROCEDURE LINE;

This PROCEDURE returns the INTEGER value of the line number which indicates the next line to be printed. Thus EJECT(LINE+3) will skip three lines and not alter spacing. After each call on OUTIMAGE, the line number is incremented by the current spacing.

PROCEDURE OUTIMAGE;

This PROCEDURE acts like the OUTIMAGE of OUTFILE but in addition increments the line number by spacing, and will position to the top of the next page if the current page is filled.

CLASS BASICIO

The system defined file facilities are grouped together in the CLASS BASICIO whose skeleton reads:

```
CLASS BASICIO(linelength); INTEGER linelength;
BEGIN  CLASS file.....................;
       file CLASS INFILE.............;
       file CLASS OUTFILE............;
       file CLASS DIRECTFILE.........;
       file CLASS PRINTFILE..........;
       REF(INFILE)sysin;
       REF(PRINTFILE)sysout;
       REF(INFILE) PROCEDURE SYSIN; SYSIN :- sysin;
       REF(PRINTFILE) PROCEDURE SYSOUT; SYSOUT :- sysout;
       sysin :- NEW INFILE("SYSIN");
       sysin.OPEN(BLANKS(80));
       sysout :- NEW PRINTFILE("SYSOUT");
       sysout.OPEN(BLANKS(linelength));
       INNER;
       sysin.CLOSE; sysout.CLOSE;
END ***BASICIO***
```

When the actions of the user defined program are exhausted, control returns to the prefix level of the BASICIO object and continues after the INNER. The following three statements close the three system generated files.

The inspect statements enclosing the program allow the user to write ININT, INIMAGE,..... instead of SYSIN.ININT, SYSIN.IMAGE and OUTREAL, OUTIMAGE,.... instead of SYSOUT.OUTREAL, SYSOUT.OUTIMAGE. There are attribute name clashes

OPEN, CLOSE } which should never be used for SYSIN or SYSOUT
IMAGE
SETPOS
POS
MORE
LENGTH

When these occur they are naturally bound to SYSOUT and the corresponding attributes of SYSIN may be obtained by writing SYSIN.SETPOS, SYSIN.IMAGE etc. Alternatively, an input section may be written as

```
INSPECT SYSIN DO
BEGIN
        input - in this block occurrences IMAGE, SETPOS,
        POS, MORE and LENGTH are bound to SYSIN
END;
```

Arithmetic functions

Certain identifiers, expressed as procedures are defined by the Simula system for standard arithmetic functions.

ABS(E)	modulus (absolute value) of E
ARCCOS(E) ARCSIN(E) ARCTAN(E)	return the principal values of the arc-cosine, arc-sine, arc-tangent of E (E is measured in radians)
COS(E) SIN(E) TAN(E)	return the cosine, sine, tangent of E (E is measured in radians)
COSH(E) SINH(E) TANH(E)	return the hyperbolic cosine, hyperbolic sine, hyperbolic tangent of E (E is measured in radians)
EXP(E)	exponential function of E (e^E)
LN(E)	natural logarithm of E ($\log_e E$, or ln E). If E <= 0, a run time error results.
SQRT(E)	returns the square root of E if E >= 0. If E < 0, a run time error results.

The above 13 functions operate on arithmetic arguments. If the type of E is [SHORT]INTEGER or REAL, then the function value is of type REAL. If the type of E is LONG REAL, then the function value is of type LONG REAL.

MOD(M,N) M modulo N, that is

M - ((M//N)*N)

e.g. MOD (7,3) is 1
MOD (-48,5) is 2

The function operates on [SHORT]INTEGER arguments, LONG REAL arguments being rounded. The result is [SHORT]INTEGER.

Utility procedures

PROCEDURE HISTO(A,B,C,D);{INTEGER / REAL} ARRAY A,B;
{INTEGER / REAL} C,D;

A call on HISTO updates a histogram defined by the one dimensional ARRAYS (INTEGER or REAL) A,B according to observation C with weight D. A(I) is incremented by D, where I is the smallest INTEGER such that C <= B(I). It is assumed that the length of A is one greater than the length of B. The last element of A corresponds to those observations which are greater than all the elements of B.

PROCEDURE LOWTEN(C); CHARACTER C;

Without use of LOWTEN, the CHARACTER 'E' represents the exponent sign in any numeric item to be edited or de-edited.
A call on "LOWTEN" with actual parameter "EXPSIGN" will replace 'E' by the value of EXPSIGN in future editing and de-editing.

SEQUENCING PROCEDURES

```
PROCEDURE CALL(X); REF(anyclass)X;
PROCEDURE DETACH;
PROCEDURE RESUME(Y); REF(anyclass)X;
```

RANDOM DRAWING PROCEDURES

```
INTEGER PROCEDURE DISCRETE(A,U); NAME U; ARRAY A; INTEGER U;
BOOLEAN PROCEDURE DRAW(A,U); NAME U; REAL A; INTEGER U;
   REAL PROCEDURE ERLANG(A,B,U); NAME U; REAL A,B; INTEGER U;
INTEGER PROCEDURE HISTD(A,U); NAME U; ARRAY A; INTEGER U;
   REAL PROCEDURE LINEAR(A,B,U); NAME U; ARRAY A,B; INTEGER U;
   REAL PROCEDURE NEGEXP(A,U); NAME U; REAL A; INTEGER U;
   REAL PROCEDURE NORMAL(A,B,U); NAME U; REAL A,B; INTEGER U;
INTEGER PROCEDURE POISSON(A,U); NAME U; REAL A; INTEGER U;
INTEGER PROCEDURE RANDINT(A,B,U); NAME U; INTEGER A,B,U;
   REAL PROCEDURE UNIFORM(A,B,U); NAME U; REAL A,B; INTEGER U;
```

UTILITY PROCEDURES

```
PROCEDURE HISTO(A,B,C,D); ARRAY A,B; REAL C,D;
PROCEDURE LOWTEN(C); CHARACTER C;
```