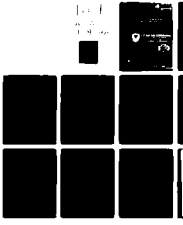
AD-A119 398 ARMY ARMAMENT RESEARCH AND DEVELOPMENT COMMAND WATER--ETC F/G 9/2
SALOME, A STRUCTURED AND LOGICALLY MINIMAL ENSEMBLE OF PROGRAMM--ETC(U)
JUL 82 R W SOANES

UNCLASSIFIED ARLCB-TR-82021 SBI-AD-E440 160 NL



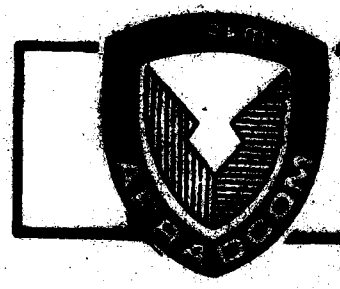
END
DATE FILMED
10-82
DTIC

TECHNICAL REPORT ARLCB-TR-82018

SALOME, A STRUCTURED AND LOGICALLY [illegible] ENSEMBLE OF PROGRAMMING CONSTRUCTS

Royce W. Soanes, Jr.

July 1982



US ARMY ARMAMENT RESEARCH AND DEVELOPMENT [illegible]
LARGE CALIBER WEAPON SYSTEMS LABORATORY
BENET WEAPONS LABORATORY
WATERVLIET, N. Y. 12189

AMCMS No. 611102H600011
DA Project No. 1L161102AH60
PRON No. 1A2250041A1A

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The use of trade name(s) and/or manufacturer(s) does not constitute an official indorsement or approval.

DESTRUCTION

Destroy this report when it is no longer needed. Do not return it to the originator.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARLCB-TR-82021	2. GOVT ACCESSION NO. AD-A119 398	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SALOME, A STRUCTURED AND LOGICALLY MINIMAL ENSEMBLE OF PROGRAMMING CONSTRUCTS		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Royce W. Soanes, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS US Army Armament Research & Development Command Benet Weapons Laboratory, DRDAR-LCB-TL Watervliet, NY 12189		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS AMCMS No. 611102H600011 DA Project No. 1L161102AH60 PRON No. 1A2250041A1A
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Armament Research & Development Command Large Caliber Weapon Systems Laboratory Dover, NJ 07801		12. REPORT DATE July 1982 13. NUMBER OF PAGES 23
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Presented at 1982 Army Numerical Analysis and Computers Conference, US Army Engineers Waterways Experiment Station, Vicksburg, MS, 3-4 February 1982. Published in proceedings of the conference.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Structured Language	Strings	Program Generator
Structured Programming	Compiler	Source Language
Language Translation	Precompiler	Target Language
Language Translator		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An overview of the Salome structured programming language will be given, along with an appendix briefly describing its syntax and semantics. Salome has been designed with the Fortran programmer in mind. The Salome source language is supported by a translator having a target language of standard Fortran, and Fortran code may be injected into Salome source code when needed. The Salome translator has been written in Salome and Fortran.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

OVERVIEW

The acronym SALOME stands for structured and logically minimal ensemble (or selection and looping operations made easy). The term "structured" refers to the gotoless nature of the logical control statements of Salome. While the power of the goto statement has corrupted many programs, the logical control statements of Salome virtually eliminate the need to use goto statements and labels in programs. The phrase "logically minimal" refers to the fact that the variety and semantic ambiguity of the structured logical control statements are kept to a minimum.

There is one looping construct and one selection construct in Salome, while other languages may force the programmer to learn several. Flexibility and unambiguous semantics are stressed in Salome - not variety.

Blanks are as important in Salome as they are in ordinary English text. The use of blanks as delimiters enhances readability, eliminates extraneous punctuation, and allows for greater brevity of syntax. Although Salome is delimiter oriented (as opposed to line oriented) there is no general end of statement or between statement delimiter in Salome. A small penalty one pays for this feature is that blanks are not allowed in assignment statements. This was considered to be a small price to pay for the elimination of a lot of extraneous semicolons.

The Salome language is supported by a one pass translator whose target language is Fortran; hence, when one writes a program in Salome, one is in effect writing in two different high level languages at the same time. The translator is in turn supported by a package of Fortran callable string manipulation routines. The primary reason Fortran was selected as the target

language was to benefit those unfortunate programmers who are inextricably mired down in a Fortran environment when Fortran is more lacking in structured programming constructs than any other high level language except Basic. A widely used high level target language also assures greater portability of Salome and its translator.

The file produced by the Salome translator consists of the original Salome source code inserted as special Fortran comments interleaved with the generated Fortran code. When Salome syntax errors are caught, they are pointed out by a string of dots. If there are syntax errors in a Salome routine, only the first one in that routine is flagged and the rest of the code in that routine is ignored. No attempt at error recovery is made because one can't be absolutely positive about what the programmer actually intended, and guessing usually uncovers syntax errors that aren't.

The reason for the interleaving of the Salome source code as comments with the generated Fortran code is to enable the programmer to relate any syntax errors picked up by the Fortran compiler (and not picked up by the Salome translator) back to the original Salome source code. The Salome translator does not check for things that the Fortran compiler is going to check for anyway.

Since the appendix of this paper contains a brief but fairly complete description of Salome, no further details will be described here. Instead, a small subroutine written in Salome along with the generated Fortran and the interleaved file are presented as a small exercise in deduction. If one knows Fortran and one reads the interleaved file carefully, one can deduce the exact semantic meaning of the Salome routine.

SALOME SOURCE EXAMPLE

```
SUB LININT ( N X Y XI YI ) -- ----------------------------------LININT
-- LINEAR INTERPOLATION
(--
N=NO. OF POINTS.
X=ABSCISSA ARRAY.
Y=ORDINATE ARRAY.
XI=ABSCISSA AT WHICH INTERPOLATION IS DESIRED.
YI=INTERPOLATED RESULT.
X IS ASSUMED TO BE SORTED IN ASCENDING ORDER.
--)
DIM X 1 , Y 1 .
-- ASSUME AT LEAST 2 POINTS OF DATA
@ N > 1 @
-- GET PROPER SUBINTERVAL USING BINARY SEARCH
IL=1 IR=N
DO # IL+1 = IR #
   -- COMPUTE INDEX OF ABSCISSA MIDWAY BETWEEN IL AND IR
   IM=(IL+IR)/2
   -- REDEFINE IL OR IR AS IM
   IF XI < X(IM) , IR=IM ; IL=IM FI
   -- ASSUME IL AND IR ARE STILL IN PROPER ORDER
   @ IL < IR @ OD
DX=X(IR)-X(IL) DY=Y(IR)-Y(IL)
-- ASSUME LENGTH OF SUBINTERVAL IS POSITIVE
@ DX > 0. @
YI=Y(IL)+(DY/DX)*(XI-X(IL))
RET END
```

FORTRAN GENERATED FROM SALOME SOURCE EXAMPLE

```
      SUBROUTINE LININT(N,X,Y,XI,YI)
C     ------------------------------------LININT
C     LINEAR INTERPOLATION
C
C     N=NO. OF POINTS.
C     X=ABSCISSA ARRAY.
C     Y=ORDINATE ARRAY.
C     XI=ABSCISSA AT WHICH INTERPOLATION IS DESIRED.
C     YI=INTERPOLATED RESULT.
C     X IS ASSUMED TO BE SORTED IN ASCENDING ORDER.
C
      DIMENSION X(1),Y(1)
C     ASSUME AT LEAST 2 POINTS OF DATA
      IF(N.GT.1) GO TO 1000
      WRITE(6,1001)
 1001 FORMAT(/////,' N > 1 IS FALSE IN LININT')
      CALL EXIT
C     GET PROPER SUBINTERVAL USING BINARY SEARCH
 1000 IL=1
      IR=N
 1002 IF(IL+1.EQ.IR) GO TO 1003
C     COMPUTE INDEX OF ABSCISSA MIDWAY BETWEEN IL AND IR
      IM=(IL+IR)/2
C     REDEFINE IL OR IR AS IM
      IF(.NOT.(XI.LT.X(IM))) GO TO 1005
      IR=IM
      GO TO 1004
 1005 IL=IM
C     ASSUME IL AND IR ARE STILL IN PROPER ORDER
 1004 IF(IL.LT.IR) GO TO 1006
      WRITE(6,1007)
 1007 FORMAT(/////,' IL < IR IS FALSE IN LININT')
      CALL EXIT
 1006 GO TO 1002
 1003 DX=X(IR)-X(IL)
      DY=Y(IR)-Y(IL)
C     ASSUME LENGTH OF SUBINTERVAL IS POSITIVE
      IF(DX.GT.0) GO TO 1008
      WRITE(6,1009)
 1009 FORMAT(/////,' DX > 0.  IS FALSE IN LININT')
      CALL EXIT
 1008 YI=Y(IL)+(DY/DX)*(XI-X(IL))
      RETURN
      END
```

INTERLEAVED SALOME SOURCE AND GENERATED FORTRAN

```
C--S--SUB LININT ( N X Y XI YI ) -- ------------------------------LININT
      SUBROUTINE LININT (N,X,Y,XI,YI)
C     ----------------------------------------LININT
C--S--  LINEAR INTERPOLATION
C     LINEAR INTERPOLATION
C--S--(--
C
C--S--N=NO. OF POINTS.
C     N=NO. OF POINTS.
C--S--X=ABSCISSA ARRAY.
C     X=ABSCISSA ARRAY.
C--S--Y=ORDINATE ARRAY.
C     Y=ORDINATE ARRAY.
C--S--XI=ABSCISSA AT WHICH INTERPOLATION IS DESIRED.
C     XI=ABSCISSA AT WHICH INTERPOLATION IS DESIRED.
C--S--YI=INTERPOLATED RESULT.
C     YI=INTERPOLATED RESULT.
C--S--X IS ASSUMED TO BE SORTED IN ASCENDING ORDER.
C     X IS ASSUMED TO BE SORTED IN ASCENDING ORDER.
C--S--)
C
C--S--DIM X 1 , Y 1 .
      DIMENSION X(1),Y(1)
C--S--  ASSUME AT LEAST 2 POINTS OF DATA
C     ASSUME AT LEAST 2 POINTS OF DATA
C--S--@ N > 1 @
      IF(N.GT.1) GO TO 1000
      WRITE(6,1001)
 1001 FORMAT(/////,' N > 1 IS FALSE IN LININT')
      CALL EXIT
C--S--  GET PROPER SUBINTERVAL USING BINARY SEARCH
C     GET PROPER SUBINTERVAL USING BINARY SEARCH
C--S--IL=1 IR=N
 1000 IL=1
      IR=N
C--S--DO # IL+1 = IR #
 1002 IF(IL+1.EQ.IR) GO TO 1003
C--S--    -- COMPUTE INDEX OF ABSCISSA MIDWAY BETWEEN IL AND IR
C     COMPUTE INDEX OF ABSCISSA MIDWAY BETWEEN IL AND IR
C--S--    IM=(IL+IR)/2
      IM=(IL+IR)/2
C--S--    -- REDEFINE IL OR IR AS IM
C     REDEFINE IL OR IR AS IM
```

```
C__S__   IF XI < X(IM) , IR=IM ; IL=IM FI
      IF(.NOT.(XI.LT.X(IM))) GO TO 1005
      IR=IM
      GO TO 1004
 1005 IL=IM
C__S__   -- ASSUME IL AND IR ARE STILL IN PROPER ORDER
C     ASSUME IL AND IR ARE STILL IN PROPER ORDER
C__S__   @ IL < IR @ OD
 1004 IF(IL.LT.IR) GO TO 1006
      WRITE(6,1007)
 1007 FORMAT(/////,' IL < IR IS FALSE IN LININT')
      CALL EXIT
 1006 GO TO 1002
C__S__DX=X(IR)-X(IL) DY=Y(IR)-Y(IL)
 1003 DX=X(IR)-X(IL)
      DY=Y(IR)-Y(IL)
C__S__-- ASSUME LENGTH OF SUBINTERVAL IS POSITIVE
C     ASSUME LENGTH OF SUBINTERVAL IS POSITIVE
C__S__@ DX > 0. @
      IF(DX.GT.0) GO TO 1008
      WRITE(6,1009)
 1009 FORMAT(/////,' DX > 0. IS FALSE IN LININT')
      CALL EXIT
C__S__YI=Y(IL)+(DY/DX)*(XI-X(IL))
 1008 YI=Y(IL)+(DY/DX)*(XI-X(IL))
C__S__RET END
      RETURN
      END
```

APPENDIX

A QUICK GUIDE TO THE SALOME PROGRAMMING LANGUAGE

1. Definition of Isolated String

   An isolated string resides on a single line, is preceded by one or more blanks (or begins in Column 1) and is followed by one or more blanks (or ends in Column 72).

   All Salome keywords must be isolated strings containing no blanks.

2. Comments

   A. A 'tack on' comment may stand alone on a line or it may be tacked onto the end of another statement.

      Ex.

```
-- TACK ON COMMENT STANDING ALONE
A=B+C -- TACKED ON TACK ON COMMENT
```

      '--' is the keyword for tack on comments

      The tack on comment has essentially the same form as the ADA comment.

   B. A delimited comment may occupy more than one line.

      Ex.

```
(-- THIS IS
A DELIMITED
COMMENT--)
```

      '(--' AND '--)' are the keywords for delimited comments.

      Delimited comments may be used to temporarily disable certain sections of executable code and they may be nested.

3. Injected Fortran

Fortran statements may be injected into a Salome program when needed.

Ex.

```
F       COMMON A(100),B(50)
F       A = B * C
```

'F' is the keyword for injecting Fortran. The Fortran statement begins immediately after the blank following 'F'.

4. Variable Declaration Statements

The correspondence between Salome variable declaration keywords and Fortran variable declaration keywords is given by the following table.

SALOME	FORTRAN
DIM	DIMENSION
INT	INTEGER
REAL	REAL
DP	DOUBLE PRECISION
LOG	LOGICAL
EQV	EQUIVALENCE

Ex.

```
DIM X 10 20 , Y 100 . translates to
DIMENSION X(10,20),Y(100)

INT A , B , C 10 20 30 .  translates to
INTEGER A,B,C(10,20,30)
```

The aforementioned keywords plus ',' and '.' are the keywords for declaration statements.

'.' Ends all variable declaration statements.

',' Separates variable references in all variable declaration statements except the equivalence statement.

',' Separates variable group references in the equivalence statement.

Ex.

```
EQV A B C , K(5) L M(10) .  Translates to
      EQUIVALENCE (A,B,C),(K(5),L,M(10))
```

Note that a string such as A(10,20) may appear in an equivalence statement but not in any other variable declaration statement.

5. Assignment Statements

Salome assignment statements must be isolated strings containing no blanks. They are otherwise equivalent to Fortran assignment statements. If blanks are deqired in assignment statements, they may be injected as Fortran.

Ex.

```
A=B*C+D P=Q*R

F       A = B + C * D / E
```

6. I/O

A. The keywords which begin I/O statements are:

'IN' 'OUT' 'R' 'W' 'RF' 'WF' 'RU' 'WU' 'FMT'

They may be read as follows:

IN - Input on device no.
OUT - Output on device no.
R - Read
W - Write
RF - Read in Format
WF - Write in Format
RU - Read Unformatted
WU - Write Unformatted
FMT - Format

B. The rest of the I/O keywords are:

'(' ')' '.' 'EOF:' 'ERR:'

'(' and ')' bracket format strings in 'R' and 'W' statements.

'.' ends all I/O statements except for the IN and OUT statements.

'EOF:' precedes any end of file label name.

'ERR:' precedes any error label name.

Ex.

```
IN 1

OUT IOUT

R I X Y ( I3 2F10.0 ) EOF: END-OF-DATA .

W I X Y ( ' I=' I3 ' X=' E14.7 ' Y=' E14.7 ) .

RF F1 A B EOF:  ENDDATA .
FMT F1 8F10.0 .

W ( ' PLAIN HOLLERITH ' ) .
```

Note that format strings in 'FMT' statements are not enclosed in parentheses as they are in 'R' and 'W' statements.

Format references in RF, WF, and FMT statements may be any isolated string containing no blanks.

7. External Subroutine Calls

External subroutines are called by writing the name of the subroutine followed by its argument list, if any. No 'call' keyword is used and English Salome keywords may not be used as subroutine names.

Ex.

EXIT - CALL TO EXIT

ADD (A B C) -- ADD A TO B GIVING C

DEFSTR (ABC ' A B C ') -- DEFINE STRING

The only keywords associated with an external subroutine call are '(' and ')'.

Arguments in an external subroutine call must be isolated strings. The only argument which may contain blanks is quoted Hollerith. Arguments are not separated by commas.

8. External Subprogram Declaration Statements

The keywords beginning external subprogram declarations and their Fortran counterparts are given by the following table.

SALOME	FORTRAN
SUB	SUBROUTINE
FUN	FUNCTION
IFUN	INTEGER FUNCTION
RFUN	REAL FUNCTION
LFUN	LOGICAL FUNCTION
DPFUN	DOUBLE PRECISION FUNCTION

'(' and ')' are the only other external subprogram declaration keywords.

Additional keywords necessary to complete the definition of an external subprogram are:

SALOME	FORTRAN
==>	ENTRY
RET	RETURN
END	END

Ex.

SUB INIT — INITIALIZATION ROUTINE

SUB MULT (A B C) — MULTIPLY B BY A GIVING C

FUN PROD (X Y) — Product of X and Y

Parameters in an external subprogram declaration must be isolated strings containing no blanks.

9. Internal Subroutine Calls

External subroutine calls in Salome are quite similar to those of Fortran, but Salome also has the facility for defining and calling internal subroutines. These sections of code are called internal because they belong to the program or subprogram which uses them and they may not be called by any other external subprogram.

An internal subroutine name may be any isolated string.

An internal subroutine is called by enclosing its name in double quotes (not two single quotes). The double quotes are the only keywords associated with the internal subroutine call.

Ex.

" READ INPUT DATA "

" INITIALIZE ARRAYS "

No arguments are passed to internal subroutines. All data in the calling program is available to the internal subroutine.

10. Internal Subroutine Declaration Statements

Internal subroutines are declared at the end of the calling program (after a return or call to exit) by writing 'TO' followed by the internal subroutine name enclosed in double quotes, followed by any code, followed by 'OT'.

Ex.

```
EXIT ( OR RET )
TO " READ INPUT DATA "
   R X Y Z ( 3F10.0 ) . OT
```

Internal subroutines may call other internal subroutines within the same program or subprogram.

Internal subroutines have exactly one exit point at 'OT'.

The Salome internal subroutine simplifies the simulation of recursion considerably.

11. If Statement

The 6 keywords involved in the If Statement are:

'IF' ',' '$' '/' ';' 'FI'

The basic functions of these keywords during translation time and run time are:

'IF' Tells the translator that the first Boolean expression is about to begin.

',' Ends a Boolean expression and selects the statement sequence following to be executed if the Boolean is true. This keyword may be read: 'then'.

'$' Ends a Boolean expression and selects the statement sequence following to be executed if the Boolean is false. This keyword may be read: 'Then Don't Select'.

'/' Halts execution in the if statement if the previous statement sequence has been executed and tells the translator that another Boolean expression is about to begin. This keyword may be read: 'Otherwise,If'.

';' Halts execution in the if statement if the previous statement sequence has been executed and selects the statement sequence following to be executed if no other statement sequence has yet been executed. This keyword may be read 'Otherwise'.

'FI' ends the If Statement.

The Relational and Boolean operators used in Boolean expressions are:

'<' '>' '<=' '>=' '=' '/=' 'AND' 'OR' 'NOT'

Relational operators must not contain blanks, but they need not be isolated. The Boolean operators 'AND', 'OR' and 'NOT' must be isolated and not contain blanks.

The general form of the If Statement is as follows:

```
IF B1 , S1 /
   B2 , S2 /
   B3 , S3 /
      .
      .
   BI , SI /
      .
      .
   BN , SN ; S FI
```

Where B1 through BN are Boolean expressions and S1 through SN and S are sequences of statements.

If Statements may be nested to theoretically any level.

Ex.

```
IF A>0. AND B>0. , X=A*B ; X=0. FI

IF I/=0 , X=0. Y=0. Z=0. FI

IF X < A , F=-1. /
   X > B , F=1. ; F=0. FI

IF I<1 , I=1 /
   I>N , I=N FI
```

12. Assertion or Assumption Statements

One may very quickly insert error checking code into a program via the assumption or assertion statement.

This statement consists of nothing more than a Boolean expression delimited on both sides by the keyword '@'.

Ex.

```
SUB INTERP ( N X Y XI YI ) -- INTERPOLATION ROUTINE
.
.
@ N>1 @
.
.
END
```

The assumption that N > 1 in subroutine interp will translate into the following typical Fortran code.

```
      IF(N.GT.1) GO TO 1000
      WRITE(6,1001)
1001  FORMAT(/////,' N>1 IS FALSE IN INTERP')
      CALL EXIT
1000  CONTINUE
```

The Salome assertion statement is considerably less verbose than the corresponding Fortran error checking code and will therefore make it far easier for the programmer to give his program the higher order of intelligence it needs in order to detect bad data or situations and subsequently notify the programmer in fairly explicit terms.

The write statement generated by the assertion statement always writes to 6.

13. Goto Statements and Labels

Although they will seldom be needed in a well written Salome program, goto statements and labels are available in Salome.

The Salome keyword '->' corresponds to the keyword "goto" in Fortran. In Fortran, labels must be numbers (statement numbers). Fortran labels therefore have no mnemonic value. In Salome, however, label names may be any isolated string containing no blanks. Salome labels may therefore be given considerable mnemonic value.

The Salome goto statement contains the goto arrow followed by a label name. The corresponding label statement or destination of the goto consists of the label name enclosed in the special brackets: '<<' and '>>'. This convention makes labels stand out well and is used in the ADA programming language. The label statement corresponds to the 'CONTINUE' statement in Fortran.

The only keywords associated with goto and label statements are:

'->' '<<' '>>'

Ex.

```
.
.
<< DATA-INPUT-SECTION >>
.
.
-> DATA-INPUT-SECTION
.
.
-> END-OF-PROGRAM
.
.
<< END-OF-PROGRAM >>
END
```

14. Loops

Salome has a single looping statement with auxiliary loop escape and loop continue statements.

A Salome loop starts with 'DO' and ends with 'OD'. Whatever code lies between these two keywords is executed over and over again until some form of loop escape is done.

The sharp sign (#) is the basic symbol used to indicate loop escapes.

To escape from 1,2,3 or 4 surrounding loops if Boolean expression B is true, one writes:

```
# B #
   or
## B ##
     or
### B ###
       or
#### B ####
```

Respectively

To escape from 1,2,3 or 4 surrounding loops unconditionally, one writes the loop escape arrows:

```
-#>
 or
--#>
  or
---#>
   or
----#>
```

Respectively

To escape from more than 4 levels of loop nesting, one must use a goto and label.

While a loop escape statement breaks off execution of a loop completely, a loop continue statement breaks off execution of a single pass or iteration and then continues execution back at the beginning of the loop. The colon (:) is the basic symbol used to indicate loop continuation.

In Fortran, when one 'GOES TO' the last statement in a 'DO' loop (often a 'CONTINUE' statement), one is doing loop continuation.

To continue 1,2,3 or 4 surrounding loops if Boolean expression B is true, one writes:

```
: B :
   or
:: B ::
     or
::: B :::
       or
:::: B ::::
```

Respectively

To continue 1,2,3 or 4 surrounding loops unconditionally, one writes the loop continue arrows:

```
<:-
 or

<:--
  or
<:---
   or
<:----
```

Respectively

To continue beyond 4 levels of loop nesting, one must use a goto and label.

The unconditional loop escape and loop continue statements allow the programmer to perform some action just prior to escape or continuation. The loop continuation constructs are seldom needed, but if they are needed, one should be careful to increment any loop index at the beginning of the loop in order to avoid an endless loop.

Ex.

```
-- MULTIPLY M X N MATRIX A TIMES N X P MATRIX B TO
-- GIVE M X P MATRIX C

-- MAKE SOME EXPLICIT ASSUMPTIONS
@ M>0 @ @ N>0 @ @ P>0 @

(-- IF THESE ASSUMPTIONS ARE NOT MADE EXPLICITLY AND ANY ONE OF THEM
TURNED OUT TO BE FALSE, NO ERROR WOULD OCCUR, BUT MATRIX C WOULD NOT BE
DEFINED AND NEITHER THE REST OF THE PROGRAM NOR THE PROGRAMMER WOULD BE
MADE IMMEDIATELY AWARE OF THIS SITUATION--)

I=0
DO I=I+1 # I>M # -- INDEX ROW OF A
   J=0
   DO J=J+1 # J>P # -- INDEX COLUMN OF B
      K=0 C(I,J)=0.
      DO K=K+1 # K>N # -- INDEX SUMMAND OF ROW A/COL B
                       -- INNER PRODUCT
         C(I,J)=C(I,J)+A(I,K)*B(K,J)
            OD OD OD -- END OF MATRIX MULTIPLICATION

-- LET ARRAY A HAVE N ELEMENTS
-- ADD UP THE POSITIVE ELEMENTS OF ARRAY A
I=0 S=0.
DO I=I+1 # I>N # : A(I)<=0. : S=S+A(I) OD

-- ADD UP THE ELEMENTS OF ARRAY A HAVING INDICES BETWEEN
-- OR INCLUDING N1 AND N2

-- SOME EXPLICIT ASSUMPTIONS THAT MIGHT BE MADE ARE:

@ N1 <= N2 @ @ N1 > 0 @ @ N2 <= N @
I=N1-1 S=0.
```

```
DO I=I+1 # I > N2 # S=S+A(I) OD
(-- IF ONE DIDN'T MAKE THE AFOREMENTIONED ASSUMPTIONS, ONE SHOULD BE
SOMEWHAT MORE CAREFUL IN SOLVING THE LAST PROBLEM --)

-- INITIALIZE LOOP INDEX AND ESCAPE CRITERION
IF N1<=N2 , I=N1-1 IMAX=N2 ; I=N2-1 IMAX=N1 FI
-- CHECK VALIDITY OF LOOP INDEX AND ESCAPE CRITERION
IF I<0 , I=0 FI IF IMAX>N , IMAX=N FI
S=0.  DO I=I+1 # I>IMAX # S=S+A(I) OD

(-- ONE CAN SEE HERE THAT ALTHOUGH THE GENERAL SOLUTION TO THE LAST
PROBLEM REQUIRED A LITTLE THOUGHT, THE PROGRAMMING WAS QUITE EASY --)
```

15. Debugging

Salome doesn't have any facility for debugging, but it does provide a mechanism for ignoring or not ignoring certain sequences of executable code at translation time.

One may prefix a '%' to a line of Salome code or one may surround multiple lines of Salome code with the delimiters '(%' and '%)'.

Salome code which is so delimited may be ignored or not ignored at translation time depending on whether the '%-Off' or '%-On' statements have been invoked, respectively. The default is %-Off.

The advantage of this facility with regard to debugging follows.

In the process of debugging, the programmer will have to insert various write statements, etc., into his program in order to ascertain where the program is going wrong. This is going to be especially true if the programmer hasn't taken advantage of the explicit assumption or assertion statement.

When a Fortran programmer thinks that he has all the bugs in his program licked, he may either (1) remove the debug statements and hope that he won't have to insert them again or (2) make a comment out of each and every debug statement and hope that he won't have to change them all back to executable code.

The Salome programmer may simply insert his debug statements with % delimiting. During debugging, he may activate all or some of these debug statements using the keywords '%-On' and '%-Off'. When a Salome programmer thinks he has all the bugs in his program licked, he may simply eliminate all '%-On' strings from his program and leave all the debug statements in place for possible future use.

Ex.

```
SUB INTERP ( N X Y XI YI ) -- INTERPOLATION ROUTINE
.
.
% W ( ' ENTERING INTERP' ) .
@ N > 1 @ -- ASSUME THAT THERE ARE AT LEAST TWO POINTS IN DATA
.
.
DX=X(I+1)-X(I)
(% IF DX = 0. , W ( ' DX=0. IN INTERP' ) EXIT /
      DX < 0. , W ( ' DX<0. IN INTERP' ) EXIT FI %)
@ DX > 0. @
D=DY/DX
.
.
% W ( ' EXITING INTERP' ) .
RET END
```

TECHNICAL REPORT INTERNAL DISTRIBUTION LIST

	NO. OF COPIES
COMMANDER	1
CHIEF, DEVELOPMENT ENGINEERING BRANCH	1
ATTN: DRDAR-LCB-DA	1
-DM	1
-DP	1
-DR	1
-DS (SYSTEMS)	1
-DS (ICAS GROUP)	1
-DC	1
CHIEF, ENGINEERING SUPPORT BRANCH	1
ATTN: DRDAR-LCB-SE	1
-SA	1
CHIEF, RESEARCH BRANCH	2
ATTN: DRDAR-LCB-RA	1
-RC	1
-RM	1
-RP	1
TECHNICAL LIBRARY ATTN: DRDAR-LCB-TL	5
TECHNICAL PUBLICATIONS & EDITING UNIT ATTN: DRDAR-LCB-TL	2
DIRECTOR, OPERATIONS DIRECTORATE	1
DIRECTOR, PROCUREMENT DIRECTORATE	1
DIRECTOR, PRODUCT ASSURANCE DIRECTORATE	1

NOTE: PLEASE NOTIFY DIRECTOR, BENET WEAPONS LABORATORY, ATTN: DRDAR-LCB-TL, OF ANY REQUIRED CHANGES.

TECHNICAL REPORT EXTERNAL DISTRIBUTION LIST

	NO. OF COPIES
ASST SEC OF THE ARMY RESEARCH & DEVELOPMENT ATTN: DEP FOR SCI & TECH THE PENTAGON WASHINGTON, D.C. 20315	1
COMMANDER US ARMY MAT DEV & READ. COMD ATTN: DRCDE 5001 EISENHOWER AVE ALEXANDRIA, VA 22333	1
COMMANDER US ARMY ARRADCOM ATTN: DRDAR-LC -LCA (PLASTICS TECH EVAL CEN) -LCE -LCM -LCS -LCW -TSS (STINFO) DOVER, NJ 07801	1 1 1 1 1 1 2
COMMANDER US ARMY ARRCOM ATTN: DRSAR-LEP-L ROCK ISLAND ARSENAL ROCK ISLAND, IL 61299	1
DIRECTOR US ARMY BALLISTIC RESEARCH LABORATORY ATTN: DRDAR-TSB-S (STINFO) ABERDEEN PROVING GROUND, MD 21005	1
COMMANDER US ARMY ELECTRONICS COMD ATTN: TECH LIB FT MONMOUTH, NJ 07703	1
COMMANDER US ARMY MOBILITY EQUIP R&D COMD ATTN: TECH LIB FT BELVOIR, VA 22060	1
COMMANDER US ARMY TANK-AUTMV R&D COMD ATTN: TECH LIB - DRDTA-UL MAT LAB - DRDTA-RK WARREN, MICHIGAN 48090	1 1
COMMANDER US MILITARY ACADEMY ATTN: CHMN, MECH ENGR DEPT WEST POINT, NY 10996	1
US ARMY MISSILE COMD REDSTONE SCIENTIFIC INFO CEN ATTN: DOCUMENTS SECT, BLDG 4484 REDSTONE ARSENAL, AL 35898	2
COMMANDER REDSTONE ARSENAL ATTN: DRSMI-RRS -RSM ALABAMA 35809	1 1
COMMANDER ROCK ISLAND ARSENAL ATTN: SARRI-ENM (MAT SCI DIV) ROCK ISLAND, IL 61299	1
COMMANDER HQ, US ARMY AVN SCH ATTN: OFC OF THE LIBRARIAN FT RUCKER, ALABAMA 36362	1
COMMANDER US ARMY FGN SCIENCE & TECH CEN ATTN: DRXST-SD 220 7TH STREET, N.E. CHARLOTTESVILLE, VA 22901	1
COMMANDER US ARMY MATERIALS & MECHANICS RESEARCH CENTER ATTN: TECH LIB - DRXMR-PL WATERTOWN, MASS 02172	2

NOTE: PLEASE NOTIFY COMMANDER, ARRADCOM, ATTN: BENET WEAPONS LABORATORY, DRDAR-LCB-TL, WATERVLIET ARSENAL, WATERVLIET, N.Y. 12189, OF ANY REQUIRED CHANGES.

TECHNICAL REPORT EXTERNAL DISTRIBUTION LIST (CONT.)

	NO. OF COPIES
COMMANDER US ARMY RESEARCH OFFICE P.O. BOX 12211 RESEARCH TRIANGLE PARK, NC 27709	1
COMMANDER US ARMY HARRY DIAMOND LAB ATTN: TECH LIB 2800 POWDER MILL ROAD ADELPHIA, MD 20783	1
DIRECTOR US ARMY INDUSTRIAL BASE ENG ACT ATTN: DRXPE-MT ROCK ISLAND, IL 61299	1
CHIEF, MATERIALS BRANCH US ARMY R&S GROUP, EUR BOX 65, FPO N.Y. 09510	1
COMMANDER NAVAL SURFACE WEAPONS CEN ATTN: CHIEF, MAT SCIENCE DIV DAHLGREN, VA 22448	1
DIRECTOR US NAVAL RESEARCH LAB ATTN: DIR, MECH DIV CODE 26-27 (DOC LIB) WASHINGTON, D.C. 20375	1 1
NASA SCIENTIFIC & TECH INFO FAC P.O. BOX 8757, ATTN: ACQ BR BALTIMORE/WASHINGTON INTL AIRPORT MARYLAND 21240	1

	NO. OF COPIES
COMMANDER DEFENSE TECHNICAL INFO CENTER ATTN: DTIA-TCA CAMERON STATION ALEXANDRIA, VA 22314	12 (2-LTD)
METALS & CERAMICS INFO CEN BATTELLE COLUMBUS LAB 505 KING AVE COLUMBUS, OHIO 43201	1
MECHANICAL PROPERTIES DATA CTR BATTELLE COLUMBUS LAB 505 KING AVE COLUMBUS, OHIO 43201	1
MATERIEL SYSTEMS ANALYSIS ACTV ATTN: DRXSY-MP ABERDEEN PROVING GROUND MARYLAND 21005	1

NOTE: PLEASE NOTIFY COMMANDER, ARRADCOM, ATTN: BENET WEAPONS LABORATORY, DRDAR-LCB-TL, WATERVLIET ARSENAL, WATERVLIET, N.Y. 12189, OF ANY REQUIRED CHANGES.


DATE
FILME
0-8