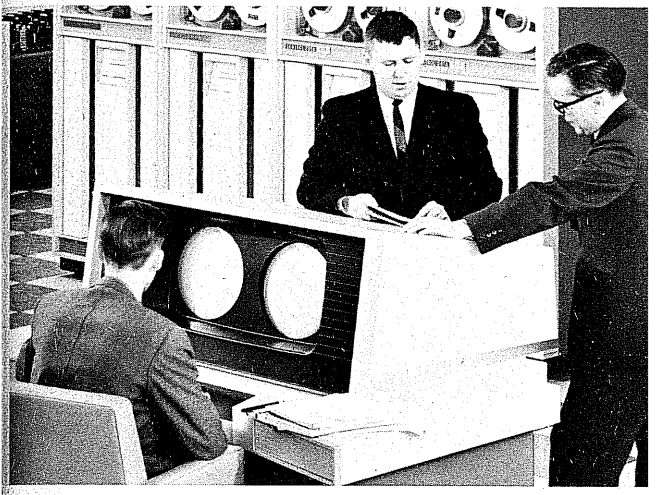
DAVID C. LEE

CONTROL DATA



INSTANT

6400/6500/6600

SIMULA

17·8·70

INSTANT 6400/6500/6600 SIMULA

CONTROL DATA® 6000 SIMULA is a dynamic general purpose programming language for algorithmic applications; the language has a powerful simulation capability. Based on ALGOL-60, SIMULA is an extension of Control Data 6000 ALGOL.

SIMULA operates under control of the SCOPE operating system, and has the following capabilities:

Simulation of discrete event systems - processes interrelated by events, dynamically evaluated, rather than by permanent relationships

Readable language

Classes - structured compound objects extend the structure concept of PL/1

Scheduling algorithms

Extensive library capabilities - tracing facilities for debugging, random drawing, data analysis and library facilities of Control Data 6000 ALGOL

Flexible input/output

List-processing

Prefixing - class names prefix program blocks and make the class capabilities available to the block

6000 SIMULA ELEMENTS

Number Format

$\pm d_1 d_2 \ldots d_i . d_{i+1} d_{i+2} \ldots d_n 10^{\pm e_1 e_2 \ldots e_m}$

real: { integer : { ±mm
$\pm_{10}{}^{\pm ee}$ $\pm .nn$ $\pm mm.nn$
$\pm mm_{10}{}^{\pm ee}$ $\pm .nn_{10}{}^{\pm ee}$ $\pm mm.nn_{10}{}^{\pm ee}$

magnitudes:

$$0 \leq \underline{\text{integer}} \leq 1.3 \times 10^{322}$$

$$3.1 \times 10^{-294} \leq \underline{\text{real}} \leq 1.3 \times 10^{322}$$

Precedence of Operators

first:	↑	sixth:	∧
second:	× /	seventh:	∨
third:	+ -	eighth:	⊃
fourth:	< ≤ = ≥ > ≠ = = =/= is, in	ninth:	≡
fifth:	¬		

Virtual Quantity Specifiers

label, switch, procedure, and <type> procedure

Blanks

Not significant except within a text or character constant.

Blocks

Maximum nesting: 32 levels

Identifiers

Maximum length: 256 characters

Specifications are required for all formal parameters; maximum number of formal parameters 63.

Procedure body may be replaced by code n.

Transmission Modes for Class Parameters and Procedure Parameters

Parameter	Transmission Mode		
	by Value	by Reference	by Name
real, integer, Bool., char.	D	////	O†
ref	////	D	O†
text	O	D	O†
real array, Boolean array, integer array, char. array	O	D	O†
ref array, text array	////	D	////
procedure	////	D†	////
label	////	D†	////
switch	////	D†	////

O Optional mode

D Default mode

//// Forbidden for class & procedure parameters transmission

† Forbidden for class parameters transmission

STANDARD PROCEDURES

Object Handling

Name	Type	
Detach	procedure	Avoid within process objects
Resume	procedure	

Set Handling (Simset)

Suc	reference procedure	
Pred	reference procedure	
Out	procedure	Dummy statement if object has no set membership
Follow	procedure	Dummy statement if object has no set membership or referenced object is none
Precede	procedure	
Into	procedure	Dummy statement if referenced object is none
First	reference procedure	
Last	reference procedure	
Empty	Boolean procedure	
Cardinal	integer procedure	
Clear	procedure	

Simulation

Idle	Boolean procedure
Terminated	Boolean procedure
Evtime	real procedure
Nextev	reference procedure
Time	real procedure
Hold	procedure
Passivate	procedure
Wait	procedure
Cancel	procedure
Current	reference procedure

Procedure names are recommended reserved names.

Utility

Name	Type
Accum	procedure

Text Handling

Sub	text procedure
Strip	text procedure
Length	integer procedure
Main	text procedure

Character Accessing

Name	Type
Pos	integer procedure
Setpos	procedure
More	Boolean procedure
Getchar	character procedure
Putchar	procedure

Editing

De-editing

Editing and De-editing operate through Putchar and Getchar

Putint	procedure	Getint	integer procedure
Putfix	procedure	Getreal	real procedure
Putreal	procedure	Getfrac	integer procedure
Putfrac	procedure		

Input/Output Procedures

Name	Type	Name	Type
Sysin	reference procedure	Sysout	reference procedure
Open	procedure	Close	procedure
Inimage	procedure	Outimage	procedure
Endfile	procedure		
Inchar	character procedure	Outchar	procedure
Inint	integer procedure	Outint	procedure
Inreal	real procedure	Outreal	procedure
Infrac	integer procedure	Outfrac	procedure
Intext	text procedure	Outtext	procedure
Lastitem	Boolean procedure	Outfix	procedure

Directfile Procedures

Locate	procedure
Location	integer procedure

Interrupt Control

Manint	procedure
Arthoflw	procedure
Parity	procedure

Printer Output

Line	integer procedure
Lines Per Page	procedure
Spacing	procedure
Eject	procedure

Standard Library Procedures

Random Drawing

Name	Type
Draw	Boolean procedure
Randint	integer procedure
Uniform	real procedure
Normal	real procedure
Psnorm	real procedure
Negexp	real procedure
Poisson	integer procedure
Erlang	real procedure
Discrete	integer procedure
Histd	integer procedure
Linear	real procedure

Character Handling

Name	Type
Rank	integer procedure
Char	character procedure
Digit	Boolean procedure
Letter	Boolean procedure

Text Generation

Blanks	text procedure
Text	text procedure

Data Analysis

Histo	procedure

Standard Functions

Abs	integer/real procedure
Sign	integer procedure
Sqrt	real procedure
Sin	real procedure
Cos	real procedure
Arctan	real procedure
Ln	real procedure
Exp	real procedure

Transfer Function

Entier	integer procedure

CLASSES AND CLASS ATTRIBUTES

Set Handling

SIMSET

LINKAGE

HEAD (subclass of LINKAGE)

LINK (subclass of LINKAGE)

Object Scheduling, Generation and Execution

SIMULATION (subclass of SIMSET)

PROCESS

Input/Output

DIRECTFILE

INFILE

OUTFILE

PRINTFILE (subclass of OUTFILE)

IMAGE (attribute of FILE)

Standard SCOPE File Names

INPUT	Standard input
OUTPUT	Standard print output
PUNCH	Standard punch output
PUNCHB	Standard binary punch output
LGO	Load-and-go (non-segmented)
SEGMENT	Segmented program
INTERM1, INTERM2, LIBRARY	Internal SIMULA compiler files

SIMULA CODING

A source deck for compilation is in the form of cards, or card images. A source deck may be a SIMULA program or a SIMULA source procedure. Source decks are stacked consecutively one behind the other, following the SCOPE control cards. The 'EOP' of the last source deck must be followed by a card containing FINIS in columns 10-14. If no compilation of a source stack is requested (G and R options), the source input stack and FINIS cards must be omitted.

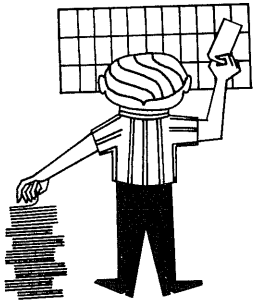
Simula Control Card

SIMULA (c_1,c_2,c_3,...,c_n)

SIMULA,c_1,c_2,c_3,...,c_n.

SIMULA.

Each parameter has the form c or c=fn where c is any sequence of 1-7 characters beginning with one of the parameter letters defined below. For example, L and LIST are equally acceptable for the list parameter. fn is the file name; if =fn is not specified, the standard file name associated with each parameter is used. Except for the I parameter, the absence of any parameter suppresses the corresponding option. If I is omitted, source input is on the standard input device.



PARAMETERS

I Source input (same as absence of I unless =fn is included). Standard file name is INPUT.

L List source input. Diagnostics are printed even if source listing is suppressed. Standard file name is OUTPUT.

X Object program in standard relocatable binary (non-segmented) load-and-go form. Standard file name is LGO.

P Object program in standard relocatable binary (non-segmented) punched form. Standard file name is PUNCHB.

S Object program in segmented form. Suppresses any X option but not P option form of the object program. Standard file name is SEGMENT. This file must be a disk file.

R Execute the object program in segmented form. If the S option is included also, the segmented program compiled is executed. If the S option is not included, the segmented program is assumed to exist already, and all options (I, U, G in particular) are suppressed. The source stack must be completely empty. Standard file name is SEGMENT. This file must be a disk file.

U User subprogram input supplementary to I. May be used only when the S option is included. Standard file name is LGO.

G User subprogram input exclusively. May be included only when the S option is included. Suppresses any explicit or implicit I option. The source stack must be empty. Standard file name is LGO. Only one of the options U or G may be included.

A List the assembly language encoded form of the object code in standard assembly language listing format. Standard file name is OUTPUT.

B Punch the assembly language encoded form of the object code in standard assembly language card format. Standard file name is PUNCH.

N Suppress array bounds checking in the object program. No file name required.

DATA SET CARDS

Data set cards appear as the first or only cards of the object-time data on the standard input device. DATASET, appears in columns 1-8. Parameters are separated by commas, blanks may appear anywhere.

Data Set Define Card

DATASET, dn=file-name, Dd, B

dn — alphanumeric string of up to 7 characters beginning with a letter. If dn is used in generation of an object of class FILE, the SCOPE file, file-name, is to be referenced.

file-name — SCOPE file-name

Dd — D2 sets density to 200 bpi, D5 to 556 bpi, D8 to 800 bpi. If omitted, or D0, density is dependent on operator or installation control

B — Binary mode, if omitted, BCD

Data Set Equate Card

DATASET, dn_1=.dn_2

dn_1 and dn_2 are character strings of up to 7 characters beginning with a letter.

Data Set End Card

DATASET,END

This card indicates the end of data set information and must be included even if there are no other data set cards in the deck.

Standard Simula Data Set Cards

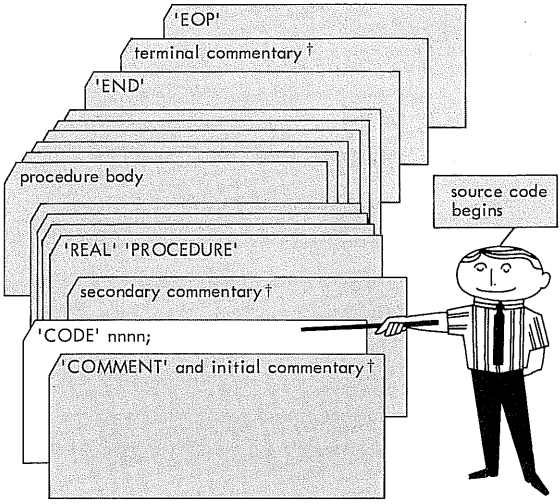
Two data set cards are supplied by the SIMULA system for SCOPE standard input and output devices.

DATASET, SYSIN = INPUT

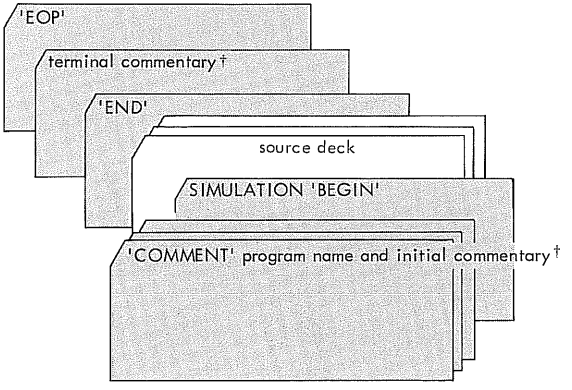
DATASET, SYSOUT = OUTPUT

The standard input/output devices may be referenced by the names SYSIN and SYSOUT and do not require data set cards.

PROCEDURE SOURCE DECK

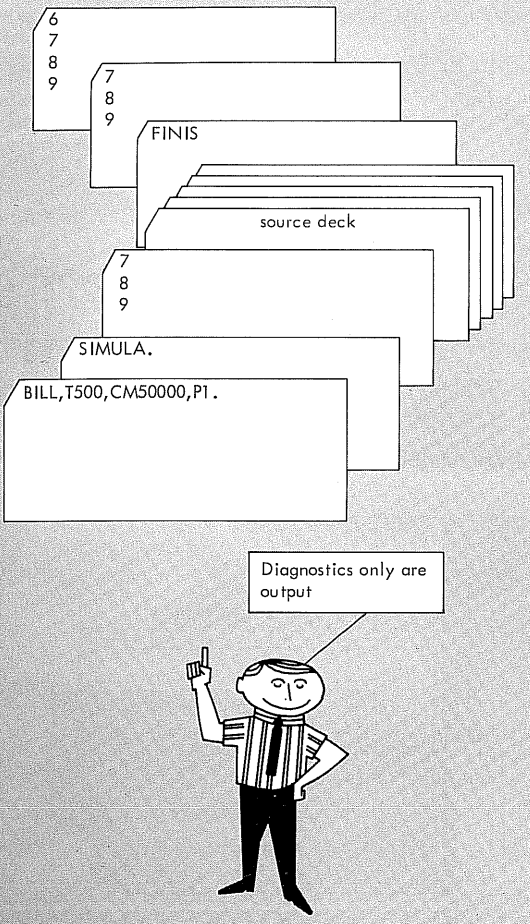


PROGRAM SOURCE DECK

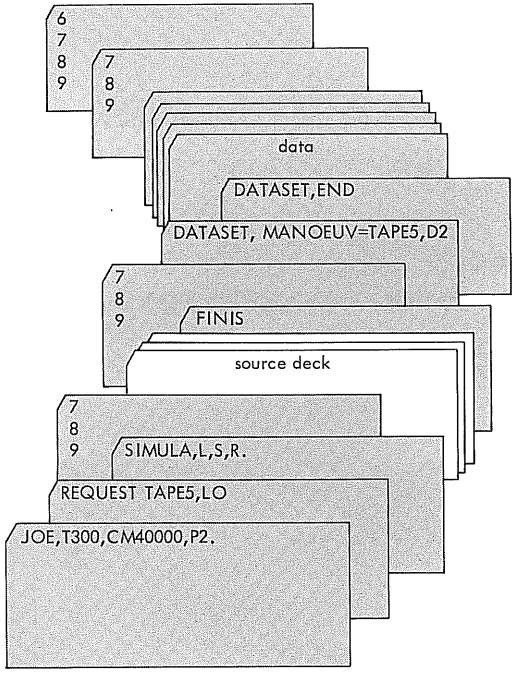


† optional

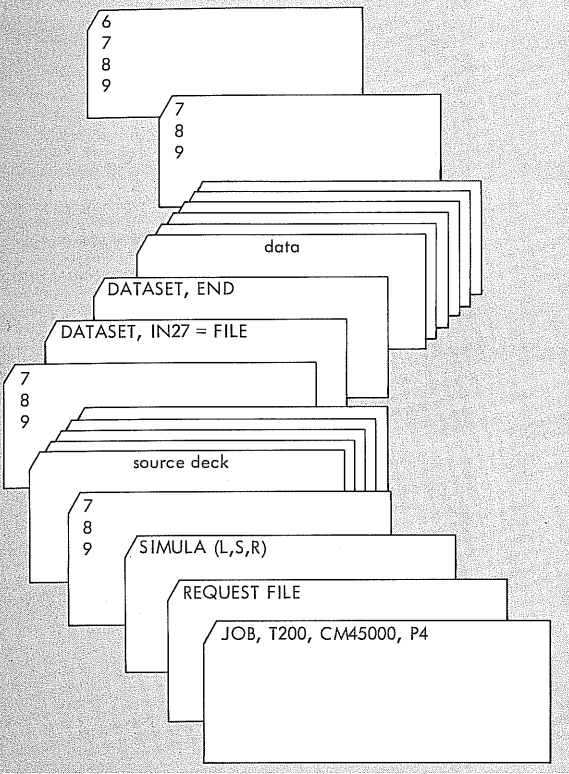
COMPILE SOURCE INPUT,LIST DIAGNOSTICS



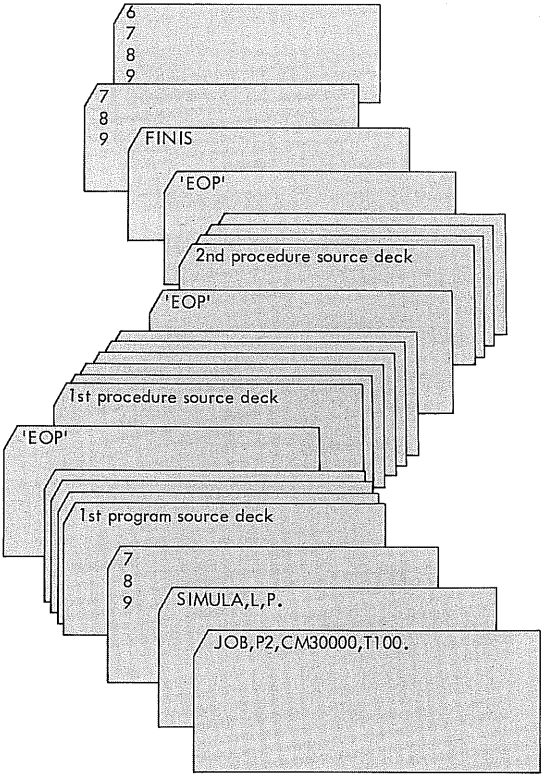
COMPILE SOURCE INPUT , LIST, EXECUTE SEGMENTED OBJECT PROGRAM



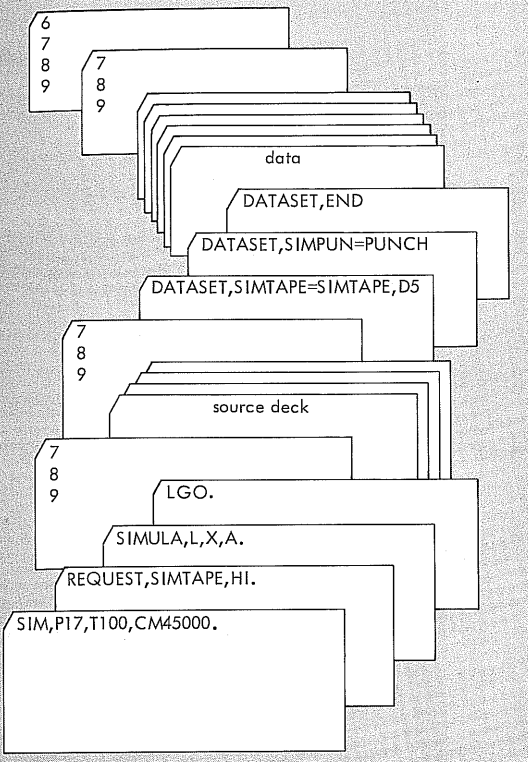
COMPILE PROGRAM TO SEGMENTED FILE, EXECUTE



COMPILE WITHOUT EXECUTION SEVERAL SOURCE DECKS; LIST AND PUNCH BINARY



COMPILE SOURCE INPUT , LIST GENERATED PROGRAM AND EXECUTE



ALGOL SYMBOL

own and string not available.

ARITHMETIC OVERFLOW

Evaluation of an expression involving constants results in an arithmetic overflow; condition is detected only if the result is subsequently used.

ARRAY BOUND TYPE

Array bound expression is not arithmetic.

ARRAY BOUND - LOCAL

Variable specified for array bound is declared at same level as array.

ARRAY OR SWITCH CALL

Identifier used as an array or switch has not been so declared.

ARRAY SIZE - NEG OR ZERO

Computed array size is negative or zero.

ARRAY, SWITCH, PROCEDURE

Too many subscripts or switch elements or formal or actual parameters.

BYPASS OVERFLOW

Capacity of compiler to handle forward references exceeded.

CALL PARAMETER

Undeclared or untyped parameter in a procedure call.

CALL PARAMETER COUNT

Procedure is called with wrong number of parameters.

CHARACTER CONSTANT

Illegal external representation of a character symbol.

'CODE' INTEGER

Literal following symbol code is not an integer.

'COMMENT'

Symbol comment in an illegal position in source text.

COMPOUND DELIMITER

Hardware representation of a SIMULA symbol is incorrect e.g., ('BIGIN').

CONFLICTING VIRTUAL

Conflicting virtual specification.

DECLARATION CAPACITY

Too many variables declared in a block structure.

DECLARATION CODE O-FLOW

Capacity of compiler to store labels, procedures, etc., for declaration code exceeded.

DELIMITER

Incorrect delimiter in source text for the particular context

DELIMITER IN COMMENT (MESSAGE)

Statement may have been bypassed because of a missing delimiter (such as ; following an end).

DELIMITER MISSING

Delimiter expected at this point in source text not found.

DOUBLE DECLARATION

Identifier declared more than once in same block heading.

DOUBLE SPECIFICATION

Formal parameter specified more than once in same procedure heading.

DOUBLE DEFINED

Two or more separately-compiled procedures with same name found during preparation of segment file.

'END'S MISSING

More begin than end symbols when 'EOP' encountered.

FINIS GEN. BY PAR. ERR EOR CARD

FINIS (end of source stack) forced at this point by parity error or EOR card in source input.

'FOR' CONTROL VARIABLE

Control variable of for statement of incorrect kind or type.

FORMAL MISSING

Value or specification appears for an identifier not in formal list.

IDENTIFIER OVERFLOW

No room in available memory to store complete list of identifiers (symbol table overflow).

'IF' CLAUSE TYPE

Expression following an if symbol must be Boolean.

'IF' EXPRESSION TYPE

Expressions following symbols then and else in if statement must be same type.

ILLEGAL CHARACTER

Illegal character (external BCD 12_8) in source text.

ILLEGAL PREFIX

Some system classes cannot be used to prefix a block.

INCOMPLETE ENTRY TABLE

Incorrect Entry Point Table.

INCOMPLETE LINK TABLE

Incorrect Linkage Table.

INCOMPLETE REPLICATION TABLE

Incorrect Replication Table.

INCOMPLETE TRANSFER TABLE

Incorrect Transfer Table.

INSTRUCTION OVERLAP

Amount of code generated by a simple arithmetic or Boolean expression exceeds the capacity of the compiler.

LABEL

Identifier used as a label not declared.

LOAD ADDRESS

Load address in text table out of range.

LOCAL VARIABLE OVERFLOW

Too many local variables defined in same block.

LONG IDENTIFIER (MESSAGE)

Identifier exceeds 256 characters.

MISSING DECLARATION

Undeclared identifier.

MISSING PROGRAM

Program appears to be missing because of absence or misspelling of a delimiter which begins compilation (e.g., begin).

MORE THAN ONE PROGRAM

More than one main program found during preparation of segment file.

NAME SPECIF

name specified for formal parameter does not permit by name transmission.

NEW SEGMENT WITHIN TEXT TABLE

Text table overlaps two segments.

NO 'CODE' INTEGER

Integer expected after symbol code is missing.

NO MAIN PROGRAM

Only code procedures found during segmentation.

NO MATCH VIRTUAL

Conflicting virtual specifications are illegal: types and kinds must coincide.

NOT SUBORDINATE

Conflicting virtual specifications are illegal: qualifications must be classes of same prefix branch.

NUMBER SIZE

Number exceeds floating-point capacity of machine.

NUMBER SYNTAX

Number incorrectly punctuated.

OPERAND

Incorrect operand in source text for the particular context.

OPERAND MISSING

Operand expected at this point in source text not found.

OPERAND OVERFLOW

Capacity of compiler to handle operands within the same statement exceeded.

PARAMETER COMMENT

Parameter comment which replaces a comma in a procedure declaration or procedure call incorrectly formed.

PREFIX LOOP

Class structures must be trees.

PREFIX NOT CLASS

Prefix of a class not a class identifier.

PROCEDURE IDENTIFIER

Identifier in a procedure call not declared as a procedure.

PROGRAM BEGINS (MESSAGE)

Line at which program compilation begins (appears with every compilation).

PROGRAM ENDS (MESSAGE)

Line at which program compilation ends (appears with every compilation).

QUALIFIER NOT CLASS

The qualification of reference variable is not a class identifier.

REDECLARATION CAPACITY

Capacity of compiler to handle similarly-spelled identifiers in a nested block structure exceeded.

REFERENCE OUTSIDE SEGMENT

Invalid addressing found during segmentation.

REPLICATION ADDRESS

Attempt to perform replication outside current segment.

REPLICATION RELOCATION

Replication may occur only within program part.

RULE ABOUT PREFIX VIOLATED

A class and its prefix must be declared within same block head.

SECOND DECLARATION

Line on which second element of DOUBLE DECLARATION is made.

SEQUENCE

Binary tables out of order.

SOURCE DECK ENDS (MESSAGE)

Line at which 'EOP' is found or forced (appears with every compilation).

SPECIFICATION MISSING

Specification missing for identifier included as a formal.

STANDARD FUNCTION PARAM

Parameter in call to standard procedure of incorrect type.

'STEP' ELEMENT TYPE

Third expression in a step element must be arithmetic.

STOP COMPILATION

Line at which compilation stops; error messages for other lines may be lost. Appears in conjunction with OPERAND OVERFLOW, SYSTEM ERROR, etc.

STRUCTURE CAPACITY

Compiler capacity to handle a nested structure, such as parenthetical statements, exceeded.

SUBPROGRAM SIZE

Current subprogram exceeds 128K words.

SUBSCRIPT TYPE

All subscript expressions must be arithmetic.

'SWITCH' PARAMETER

All elements in a switch list must be labels or designational expressions.

SYSTEM ERROR

Compiler or machine malfunction.

TERMINATION

Language construction in source text terminates illegally.

TEXT CHARACTER

Illegal character or character constant in a text (external BCD 12_8).

TEXT LENGTH

Too many characters in text or 'EOP' encountered before end of text.

TEXT TERMINATION

'EOP' encountered before end of text or character.

TOO MANY BLOCK LEVELS

A block structure contains block nested to more than 32 levels.

TOO MANY IDENTIFIERS

Too many differently spelled identifiers in the program.

TOO MANY WORKING LOGS

Too many working locations in excess of declared variables required to perform operations specified in this block.

TYPE

In a general expression, elements must have same types.

UNKNOWN PREFIX

No class identifier declared with that prefix name.

UNKNOWN QUALIFIER

Qualification of a reference variable not defined.

UNDEFINED

No external name found during preparation of segment file.

VALUE SPECIFICATION

value applied for formal parameter whose specification does not permit a value (e.g., label).

'WHILE' ELEMENT TYPE

Second expression in a while statement of illegal type.

OBJECT-TIME DIAGNOSTICS

Upon normal termination, open files are closed and the following message is output to the standard output device:

END OF SIMULA RUN

Upon abnormal termination, a diagnostic is printed on the standard output device, open files are closed, and a structured dump of information relevant to the currently active object is output. The structured dump traces the execution path through the blocks in the block structure currently active when the error occurred. Information relevant to the object is selected from core and printed in the dump as follows:

```
THIS ERROR OCCURRED AFTER          LINE xxxx
IN THE BLOCK ENTERED AT            LINE xxxx
(global information)
(environmental information)
THIS BLOCK WAS CALLED FROM         LINE xxxx
IN THE BLOCK ENTERED AT            LINE xxxx
(environmental information)
THIS BLOCK WAS CALLED FROM         LINE xxxx
IN THE BLOCK ENTERED AT            LINE xxxx
(environmental information)
```

xxxx is the line number assigned each source image line during compilation. If the block entered is a standard procedure, the word STAN appears instead of the line number.

ALPHA FORMAT ERROR

Output value too large.

ARITHMETIC OVERFLOW

Evaluation of an expression results in arithmetic overflow (e.g., division by zero) for which no provision has been made with ARTHOFLW procedure.

ARRAY BOUNDS ERROR

Computed element address in an array not within total array boundaries.

ARRAY DECLARE ERROR

Computed array size is negative or zero.

ARRAY DIMENSION ERROR

Actual parameter in a procedure call and formal in procedure declaration.

DATA SET

Defines data set on which preceding error occurred.

DATASET CARD SYNTAX CIRCULAR PARITY EOF

Either syntax of data set card is wrong or define and equate cards result in a circular definition of a data set; or an uncorrectable parity error or EOF card occurred during reading of data set cards. The incorrect card is output before the program is terminated.

DISPLAY EXCEEDED

Block structure nested to more than 32 levels; only calls to separately compiled procedures can cause this error.

END INPUT

Attempt to read a file after encountering an end-of-file.

EXPONENTIAL ERROR

Argument of EXP procedure is too large.

ILLEGAL GO TO

go to leading to another object, connected or detached, the same quasi-parallel system.

ILLEGAL IN-OUT

Illegal operation requested for equipment selected.

INSTANTANEOUS QUALIFICATION

The attribute does not belong to the class mentioned or to one of its subclasses.

LOCAL REFERENCE

A local reference to an instance of a prefixed block is forbidden.

LOGARITHM ERROR

Argument to LN procedure may not be negative or zero.

OBJECT NONE

Attempt to use an attribute of an object with a reference pointer to none.

PARAMETER COUNT ERROR

Incorrect number of actual parameters in procedure call.

PARAMETER KIND ERROR

Actual and formal parameters in procedure not the same kind.

PARAMETER TYPE ERROR

Actual and formal parameters in procedure call not the sam type.

RESUME - NONE

Object of a resume statement does not exist.

RESUME - NOT DETACHED

Parameter of a resume statement must be a detached object

RESUME - TERMINATED

The object of a resume statement is terminated.

SIMULATION - XX -

Process implied in procedure XX does not exist within the sequencing set.

SIN - COS ERROR

Argument to SIN or COS procedure is too large.

SQUARE ROOT ERROR

Argument to the SQRT procedure may not be negative.

STORAGE OVERFLOW

No more space available.

TEXT ELEMENT ERROR

Rules of TEXT ELEMENT violated.

UNASSIGNED DATASET

No data set defined for a file name used in program.

UNDEFINED FOR LABEL

Attempt to jump into middle of for statement.

VIRTUAL LABEL

Go to a virtual label not closed.

VIRTUAL PROCEDURE

Call on virtual procedure not closed.

COMPILE-TIME AND OBJECT-TIME INPUT/OUTPUT DIAGNOSTICS

System diagnostics concerning input/output usage at compile-time and object-time appear in the DAYFILE.

SIMULA-I/O-ERROR xxx ON FILE file-name

xxx takes the following values:

In general, the following values of xxx result from system error, improper use of the input/output system in a handwritten procedure, or use of the wrong segment file.

FC1	Illegal function code to input/output	object-time or compile-time
FC2	Error on call for open	
FC3	Error on call for close	
FC4	Error on reading or writing	
SG1	Error in the segment file	object-time only
SG2	Library routine missing from segment file.	
INT	Error in compiler pass input/output	compile-time only

SIMULA 67 SYNTAX

*A star indicates a rule which is not part of the ALGOL-60 Report.

The Backus normal form is used, increased by three meta-operators:

{ }	enclose a group of meta expressions forming a new meta expression
⌈ ⌉	enclose an optional group and equivalent to <empty>|{ }
...	denote a repetition of the preceding group or optional group

<basic symbol> ::= <letter>|<digit>|
<logical value>|<delimiter>

<letter> ::= A|B|C|D|E|F|G|H|I|J|K|
L|M|N|O|P|Q|R|S|T|U|V|
W|X|Y|Z|a|b|c|d|e|f|g|h|
i|j|k|l|m|n|o|p|q|r|s|t|u|
v|w|x|y|z|

<digit> ::= 0|1|2|3|4|5|6|7|8|9|

<logical value> ::= true|false

<delimiter> ::= <operator>|<separator>|
<bracket>|<declarator>|
<specificator>

<operator> ::= <arithmetic operator>|
<relational operator>|
<logical operator>|
<sequential operator>|

<arithmetic operator> ::= +|-|×|/|÷|↑

<relational operator> ::= <Algol relational operator>|
<Simula relational operator>

<Algol relational operator> ::= <|≤|=|≥|>|≠

*<Simula relational operator> ::= is|in|==|=/=

<logical operator> ::= ≡|⊃|∨|∧|¬

<sequential operator> ::= <Algol sequential operator>|<Simula sequential operator>

<Algol sequential operator> ::= go to|if|then|else|for|do|code

*<Simula sequential operator> ::= inspect|inner|new|this|qua|activate|reactivate|otherwise|delay|at|before|after|prior

<separator> ::= ,|.|10|:|;|:=|:-|step|until|while|when|comment

<bracket> ::= (|)|[|]|'|"|begin|end

<declarator> ::= <Algol declarator>|<Simula declarator>

<Algol declarator> ::= Boolean|integer|real|array|switch|procedure

*<Simula declarator> ::= character|text|class|ref (<CLASS identifier>)|virtual:

*<specificator> ::= label|value|name

<identifier> ::= <letter>|<identifier><letter>|<identifier><digit>

*<Simula identifier> ::= <identifier>|<remote identifier>

*<remote identifier> ::= <simple reference expression>.<identifier>|<simple text expression>.<identifier>

<number> ::= [+|-]<unsigned number>

<unsigned number> ::= <decimal number>|
<exponent part>|<decimal
number> <exponent part>

<decimal number> ::= <unsigned integer>|
<decimal fraction>|
<unsigned integer> <decimal
fraction>

<exponent part> ::= 10 <integer>

<decimal fraction> ::= .<unsigned integer>

<integer> ::= [+|-]<unsigned integer>

<unsigned integer> ::= {<digit>}...

*<text constant> ::= '{<character WHERE '
WRITTEN AS''>}...'

*<character constant> ::= "<character>"

*<character> ::= <ANY CHARACTER
AVAILABLE, SPACE
INCLUDED>

<variable> ::= <simple variable>|
<subscripted variable>

<simple variable> ::= <variable identifier>

*<variable identifier> ::= <Simula identifier>

<subscripted variable> ::= <array identifier>
[<subscript list>]

*<array identifier> ::= <Simula identifier>

<subscript list> ::= <subscript expression>|
<subscript list>,
<subscript expression>

<subscript expression> ::= <arithmetic expression>

*<expression> ::= <arithmetic expression>|
<Boolean expression>|
<designational expression>|
<reference expression>|
<text expression>|
<character expression>|
<text value>

*<Simula function designator> ::= <arithmetic function designator> | <Boolean function designator> | <reference function designator> | <character function designator> | <text function designator>

<arithmetic function designator> ::= <arithmetic procedure identifier>┼<actual parameter part>┼

<arithmetic procedure identifier> ::= <Simula ARITHMETIC identifier>

<actual parameter part> ::= (<actual parameter list>)

<actual parameter list> ::= <actual parameter> | <actual parameter list> <parameter delimiter> <actual parameter>

<parameter delimiter> ::= , |) <letter string> : (

<letter string> ::= <letter> | <letter string> <letter>

<actual parameter> ::= <expression> | <array identifier> | <switch identifier> | <procedure identifier>

<Boolean function designator> ::= <Boolean procedure identifier>┼<actual parameter part>┼

*<reference function designator> ::= <reference procedure identifier>┼<actual parameter part>┼

*<character function designator> ::= <character procedure identifier>┼<actual parameter part>┼

*<text function designator> ::= <text procedure identifier> ┼<actual parameter part>┼

*<Boolean procedure identifier> ::= <Simula BOOLEAN identifier>

*<reference procedure identifier> ::= <Simula REFERENCE identifier>

*<character procedure identifier> ::= <Simula CHARACTER identifier>

*<text procedure identifier> ::= <Simula TEXT identifier>

*<procedure identifier> ::= <Simula PROCEDURE identifier>

<arithmetic expression> ::= <simple arithmetic expression>|<if clause> <simple arithmetic expression> [else <arithmetic expression>]

<simple arithmetic expression> ::= <term>|<adding operator> <term>|<simple arithmetic expression> <adding operator> <term>

<term> ::= <factor>|<term> <multiplying operator> <factor>

<factor> ::= <primary>|<factor>↑ <primary>

<primary> ::= <unsigned number>| <ARITHMETIC variable>| <arithmetic function designator>|(<arithmetic expression>)

<multiplying operator> ::= X|/|÷

<adding operator> ::= +|-

<if clause> ::= if <Boolean expression> then

<Boolean expression> ::= <simple Boolean>|<if clause> <simple Boolean> [else <Boolean expression>]

<simple Boolean> ::= <implication>|<simple Boolean>≡<implication>

<implication> ::= <Boolean term>|<implication>⊃<Boolean term>

<Boolean term> ::= <Boolean factor> |
<Boolean term>∨<Boolean factor>

<Boolean factor> ::= <Boolean secondary> |
<Boolean factor>∧<Boolean secondary>

<Boolean secondary> ::= {¬} <Boolean primary>

<Boolean primary> ::= <logical value> |
<BOOLEAN variable> |
<Boolean function designator> | <relation> |
(<Boolean expression>)

*<relation> ::= <simple arithmetic expression> <Algol relational operator> <simple arithmetic expression> | <simple character expression>{< | ≤ | = | ≥ | > | ≠} <simple character expression> <simple text value> {< | ≤ | = | ≥ | > | ≠} <simple text value> | <simple text expression>{== | =/=} <simple text expression> | <simple reference expression>{{== | =/=} <simple reference expression> | { is | in} <class identifier>}

<designational expression> ::= <simple designational expression> | <if clause> <simple designational expression> {else <designational expression>}

<simple designational expression> ::= <label> | <switch designator> | (<designational expression>)

<switch designator> ::= <switch identifier> [<subscript expression>]

<switch identifier> ::= <SWITCH identifier>

<label> ::= <LABEL identifier>

*<reference expression> ::= <simple reference expression>|<if clause> <simple reference expression> else <reference expression>

*<simple reference expression> ::= none|<REF variable>|<reference function designator>|<generating reference>|<local reference>|<qualified reference>|(<reference expression>)

*<generating reference> ::= new <CLASS identifier> [<actual parameter part>]

*<local reference> ::= this <CLASS identifier>

*<qualified reference> ::= <simple reference expression> qua <CLASS identifier>

*<text expression> ::= <simple text expression>|<if clause> <simple text expression> [else <text expression>]

*<simple text expression> ::= notext|<TEXT variable>|<text function designator>(<text expression>)

*<text value> ::= <simple text value>|<if clause> <simple text value> [else <text value>]

*<simple text value> ::= <simple text expression>|<text constant>

*<character expression> ::= <simple character expression>|<if clause> <simple character expression> [else <character expression>]

*<simple character expression> ::= <character constant>|<CHARACTER variable>|<character function designator>|(<character expression>)

*<statement> ::= <Algol statement>|<Simula statement>

<Algol statement> ::= <unconditional statement> | <conditional statement>

<unconditional statement> ::= <basic statement> | <for statement> | <compound statement> | <block>

<basic statement> ::= {<label>:}...{<assignment statement> | <goto statement> | <procedure statement>}

<compound statement> ::= {<label>:}... begin <compound tail>

<compound tail> ::= <statement> end | <statement>; <compound tail>

*<block> ::= {<label>:}...{<block prefix>}<block head>; <compound tail>

*<block prefix> ::= <class identifier> <actual parameter part>

<block head> ::= begin <declaration> | <block head>; <declaration>

*<Simula statement> ::= <connection statement> | <scheduling statement> | <denotation statement>

<assignment statement> ::= {{<ARITH. variable> | <ARITH. PROCEDURE identifier>}:=}... <arithmetic expression> |

{{<BOOLEAN variable> | <BOOLEAN PROCEDURE identifier>}:=}... <Boolean expression> |

{{<CHARACTER variable> | <CHARACTER PROCEDURE identifier>}:=}... <character expression> |

{{<TEXT variable> | <TEXT PROCEDURE identifier> | <text function designator>}:=}... <text value>

<go to statement> ::= go to <designational expression>

<procedure statement> ::= <procedure identifier> <actual parameter part>

*<procedure identifier> ::= <Simula identifier>

<actual parameter part> ::= { (<actual parameter> {<parameter delimiter> <actual parameter>}...)}

<parameter delimiter> ::= , |) <letter string> :(

<letter string> ::= <letter>...

<actual parameter> ::= <expression> | <array identifier> | <switch identifier> | <procedure identifier>

<arith. for list element> ::= <arithmetic expression> {while <Boolean expression>} | <arithmetic expression> step <arithmetic expression> until <arithmetic expression>

<ref. for list element> ::= <reference expression> {while <Boolean expression>}

<text for list element> ::= <text expression> {while <Boolean expression>}

<character for list element> ::= <character expression> {while <Boolean expression>}

<arith. for clause> ::= for <ARITH. variable> := <arith. for list element> {, <arith. for list element>}...

<ref. for clause> ::= for <REF. variable> :- <ref. for list element> {, <ref. for list element>}...

<text for clause> ::= for <TEXT variable> :- := <text for list elements> {, <text for list element>}...

<character for clause> ::= for <CHAR. variable> := <character for list element> {, <character for list element>}...

<for clause> ::= <arith. for clause>|<ref. for clause>|<text for clause>|<character for clause>

<for statement> ::= {<label>:}... <for clause> <statement>

*<conditional statement> ::= {<label>:}... <if clause> <unconditional Simula statement>{else <statement>}

<if clause> ::= if <Boolean expression> then

*<unconditional Simula statement> ::= <unconditional statement>|<Simula statement>

*<connection statement> ::= inspect <reference expression> {do <statement>|{when <CLASS identifier> do <statement>} ...} {otherwise <statement>}

*<scheduling statement> ::= <activation clause> <scheduling clause>

*<activation clause> ::= {activate|reactivate} <reference expression>

*<scheduling clause> ::= {{before|after} <reference expression>|{at|delay} <arithmetic expression> {prior}}

*<denotation statement> ::= {{<REF. variable>|<REF PROCEDURE identifier>} :-}... <reference expression>
{{ <TEXT variable>|<TEXT PROCEDURE identifier>} :-} ... <text expression>

*<declaration> ::= <type declaration>|<array declaration>|<switch declaration>|<procedure declaration>|<class declaration>

<type declaration> ::= <type> <simple Algol variable> {, <simple Algol variable>}...

*<type> ::= real | integer | Boolean | character | text | ref (<CLASS identifier>)

<simple Algol variable> ::= <VARIABLE identifier>

<array declaration> ::= {<type>} array <array list>

<array list> ::= <array segment> | <array list>, <array segment>

<array segment> ::= { <ARRAY identifier>, } <ARRAY identifier> [<bound pair list>]

<bound pair list> ::= <arithmetic expression> : <arithmetic expression> { , <arithmetic expression> : <arithmetic expression>}...

<switch declaration> ::= switch <SWITCH identifier> := <designational expression> { , <designational expression>} ...

<procedure declaration> ::= <type> procedure <procedure heading> <procedure body>

<procedure body> ::= <statement> | code <integer> <CODE BODY>

<procedure heading> ::= <PROCEDURE identifier> {{ <formal parameter part> ; { <mode part>} <specification part>} | ; }

<formal parameter part> ::= (<formal parameter> { <parameter delimiter> <formal parameter>}...)

<formal parameter> ::= <identifier>

*<value part> ::= value <identifier> { , <identifier>}... ;

<specification part> ::= { <specifier> <identifier> [, <identifier>]...;} ...

<specifier> ::= <type> | [<type>] array | label | [<type>] procedure | switch

<name part> ::= name <identifier> [, <identifier>]...;

<mode part> ::= <value part> [<name part>] | <name part> [<value part>]

*<class declaration> ::= [<prefix>] <main part>

*<prefix> ::= <CLASS identifier>

*<main part> ::= class <CLASS identifier>
* {{<formal parameter part> ;
[<value part>] <specification
part>} | ;}
* [virtual : <virtual specifica-
tion part>]
* {<statement> | <split body> }

*<virtual specification part> ::= {<virtual specifier>
<identifier>
[<identifier>]...} ...

*<virtual specifier> ::= [<type>] procedure | label |
switch

<split body> ::= begin [<declaration> ;]...
[<statement> ;]...
inner
[; <statement>]... end

CHARACTER REPRESENTATION OF SIMULA SYMBOLS

SIMULA symbol	Recommended keypunch	Tolerated keypunch	Punch for recommended keypunch
A - Z	A - Z		12-1 - 0-9
a - z			
0 - 9	0 - 9		0 - 9
+	+		12
-	-		11
X	*		11-8-4
/	/		0-1
↑	↑	'POWER'	11-8-5
÷	:/	'DIV' or '/'	8-2, 0-1
>	>	'GREATER'	11-8-7
≥	≥	'NOT LESS'	12-8-5
=	=	'EQUAL'	8-3
≠	¬=	'NOT EQUAL'	12-8-6, 8-3
≤	≤	'NOT GREATER'	8-5
<	<	'LESS'	12-0
∧	∧	'AND'	0-8-7
∨	∨	'OR'	11-0
≡	≡	'EQUIV'	0-8-6
==	==	'IDENT'	8-3, 8-3
¬	¬	'NOT'	12-8-6
=/=	=/=	'NOT IDENT'	8-3, 0-1, 8-3
⊃	→	'IMPL'	0-8-5
.	.		12-8-3
, (comma)	,		0-8-3
:	:	..	8-2
;	;	.,	12-8-7
10	' (apostr.)		8-4
(	(		0-8-4
)	)		12-8-4
[	[	(/	8-7
]	]	/)	12-8-4
:=	:=	..= or .=	8-2, 8-3
:-	:-	..- or .-	8-2, 11
'	↓	' '	11-8-6
"	$		11-8-3
(blank)	space		

SIMULA CHARACTER SET

VALUE				VALUE			
octal	decimal	character	Card Punch	octal	decimal	character	Card Punch
00	0			40	32	5	5
01	1	A	12-1	41	33	6	6
02	2	B	12-2	42	34	7	7
03	3	C	12-3	43	35	8	8
04	4	D	12-4	44	36	9	9
05	5	E	12-5	45	37	+	12
06	6	F	12-6	46	38	-	11
07	7	G	12-7	47	39	*	11-8-4
10	8	H	12-8	50	40	/	0-1
11	9	I	12-9	51	41	(	0-8-4
12	10	J	11-1	52	42	)	12-8-4
13	11	K	11-2	53	43	$	11-8-3
14	12	L	11-3	54	44	=	8-3
15	13	M	11-4	55	45	blank	space
16	14	N	11-5	56	46	,	0-8-3
17	15	O	11-6	57	47	.	12-8-3
20	16	P	11-7	60	48	≡	0-8-6
21	17	Q	11-8	61	49	[	8-7
22	18	R	11-9	62	50	]	0-8-2
23	19	S	0-2	63	51	:	8-2
24	20	T	0-3	64	52	'	8-4
25	21	U	0-4	65	53	→	0-8-5
26	22	V	0-5	66	54	∨	11-0
27	23	W	0-6	67	55	∧	0-8-7
30	24	X	0-7	70	56	↑	11-8-5
31	25	Y	0-8	71	57	↓	11-8-6
32	26	Z	0-9	72	58	<	12-0
33	27	0	0	73	59	>	11-8-7
34	28	1	1	74	60	≤	8-5
35	29	2	2	75	61	≥	12-8-5
36	30	3	3	76	62	¬	12-8-6
37	31	4	4	77	63	;	12-8-7

6000 REPRESENTATION OF SIMULA SYMBOLS

SIMULA symbol	6000-Representation	SIMULA symbol	6000-Representation
activate	'ACTIVATE'	is	'IS'
after	'AFTER'	label	'LABEL'
array	'ARRAY'	name	'NAME'
at	'AT'	new	'NEW'
before	'BEFORE'	none	'NONE'
begin	'BEGIN'	notext	'NOTEXT'
Boolean	'BOOLEAN'	otherwise	'OTHERWISE'
character	'CHARACTER'	prior	'PRIOR'
class	'CLASS'	procedure	'PROCEDURE'
code	'CODE'	qua	'QUA'
comment	'COMMENT'	reactivate	'REACTIVATE'
delay	'DELAY'	real	'REAL'
delimiter	'DELIMITER'	ref	'REF'
do	'DO'	step	'STEP'
else	'ELSE'	switch	'SWITCH'
end	'END'	text	'TEXT'
false	'FALSE'	then	'THEN'
for	'FOR'	this	'THIS'
go to	'GO TO'	true	'TRUE'
if	'IF'	until	'UNTIL'
in	'IN'	value	'VALUE'
inner	'INNER'	virtual	'VIRTUAL'
inspect	'INSPECT'	when	'WHEN'
integer	'INTEGER'	while	'WHILE'

NOTES

CONTROL DATA
CORPORATION

Documentation Department

3145 Porter Drive
Palo Alto, California 94304

60235100A Printed in U.S.A.
FEBRUARY 1969