The SNOBOL4 Reference Manual is currently in preparation by the Systems Staff of the Computer Center. The attached sections (1.3, 1.5 and 2.0-2.7) are being distributed prior to the completion of the manual to provide users of SNOBOL4 with information essential to the proper running of a program but as yet unofficial and therefore subject to possible change without notice. A tentative Table of Contents and Introduction are also supplied as an indication of the structure of the manual.

Computer Center
University of California
Berkeley
April 1969
Table of Contents

Introduction

1  SNOBOL Programming Language

1.1 Named Entities

1.1.1 Syntax
1.1.2 Variables
1.1.3 Procedures
1.1.4 Labels

1.2 Basic Data Types and Operations Upon Them

1.2.1 Strings
1.2.2 Integers
1.2.3 Reals
   Numerical Comparison
1.2.4 Arrays
1.2.5 Programmer-defined Data Types and Structures
1.2.6 Names
   Indirectness Operator
   Unary Name Operator
   Name-Function
1.2.7 Patterns
   Concatenation and Alternation
   Assignment and Replacement Statements
   Association for Assignment
   Deferred Evaluation
   Pattern Matching
   Standard Procedures for Pattern Matching
1.2.8 Standard Procedures DATATYPE, IDENT, DIFFER, etc.

1.3 Input/Output for a SNOBOL4 Program

1.3.1 Standard Attachments
   Input
   Printed Output
1.3.2 Programmer-defined I/O Attachments
   Input
   Output
1.3.3 Other Standard I/O Procedures

1.4 Structure of a SNOBOL4 Program

1.5 Syntax
Table of Contents (Continued)

2 CDC 6400 Machine Dependencies

2.1 Deck Structure

2.2 Control Card

2.3 File-set Names

2.4 End of Group Mark

2.5 Source Program

2.6 Character Set

2.7 FREEZing Programs
Introduction

The SNOBOL4 Programming Language was developed by R.E. Griswold, J.F. Poage, and I.P. Polinsky (1). The basic data element or type in the language is a string of characters. Facilities are provided for manipulating strings as well as for examining them for particular patterns of characters. Manipulating facilities include joining two strings together, and breaking up a string into one or more parts. Examination of a string for patterns may range from a simple search for a given combination of characters to a complex check for any one or several of various alternatives. By combining pattern matching and manipulation facilities, portions of strings may be deleted or replaced by other strings.

A limited range of numerical capabilities for both integer and real numbers has also been incorporated into SNOBOL4. The basic operations of addition, subtraction, multiplication and division are defined for both integers and real. For integers, exponentiation is also defined. Mixed-mode arithmetic is not permitted.

SNOBOL4 provides a variety of functions called standard procedures to facilitate arithmetic operations and pattern searching techniques which are often used and/or are too complex to be programmed easily. In addition a standard procedure is available to set up the data element called an array in which a group of items is associated with one variable name through numeric indexing. Further standard procedures permit the definition and redefinition of programmer-defined procedures and even programmer-defined data structures, e.g. trees and complex numbers.

This document is intended to serve as a reference manual for the SNOBOL4 Programming Language as implemented by Charles Simonyi and Paul McJones for the CDC 6400 computer at the Computer Center of the University of California at Berkeley. It is not intended as a tutorial manual and therefore does not treat in detail the various ramifications of the

---

more powerful SNOBOL constructions. Illustrative examples are intended mainly to clarify points in the text. The first section details the elements available for writing a SNOBOL program and indicates the few constructions which are unacceptable to the SNOBOL compiler. Only compiler considerations and not machine dependencies are treated in this section. The second section deals specifically with how to run a SNOBOL4 program on the CDC 6400 computer at CAL.
1.3 Input and Output for a SNOBOL4 Program

Input and output operations are performed by attaching variables to external storage media. A variable may be attached in either of two senses, input (read) or output (write), to the card reader, card punch, line printer, magnetic tape or disk. Each time information is to be read by the program, the use of a variable attached in the input sense causes a line (a string) to be read from the corresponding file-set and to become the new value of the variable. Similarly, whenever information is to be written, the assignment of a string to a variable attached in the output sense causes the string to be written as a line on the corresponding file-set. Therefore Input and Output operations are accomplished without any explicit I/O statements. The SNOBOL4 language provides two standard variable-file-set attachments and additionally provides two standard procedures for establishing programmer-defined attachments.

1.3.1 Standard Attachments

Input: The variable INPUT is attached to the standard INPUT file-set consisting of information read in from the card reader. Whenever it is referenced, a card image is read from the input stream and becomes its new value. All eighty columns including trailing blanks of the card image comprise the eighty-character string which is the value of INPUT. Since each use of INPUT reads a card image, previous values of INPUT are lost unless they are assigned to other variables.

If a group mark (EOR) is read, the statement performing the read fails, thus enabling the program to detect such marks (see Section 2.4). If more information follows on the file, subsequent statements containing references to INPUT will not fail.

Examples:

NAME = INPUT

The string corresponding to the next card image is assigned as the value of the variable NAME.

N = 1
LOOP DATA[N] = INPUT : F(START)
  I = I + 1 : (LOOP)
START
Card images are read into the array DATA until a group mark causes the statement to fail and control to branch to the body of the program beginning at START.

Printed Output: The variable OUTPUT is attached to the standard OUTPUT file-set which contains information to be printed on the line printer after job termination. Therefore, whenever OUTPUT is given a value, which must be a string, printout is generated.

Output is printed 132 characters per line with as many lines as necessary being generated. The null string is treated as a blank character and produces a blank line.

Examples:

```plaintext
OUTPUT = 'THE COMPUTED RESULTS ARE'
```

will print

```plaintext
THE COMPUTED RESULTS ARE
```

```plaintext
EXP = EXPl . OUTPUT '+' EXP2 . OUTPUT
```

will print the two terms being assigned to EXPl and EXP2 as values when EXP is used in a pattern match. This technique is useful for tracing.

1.3.2 Programmer-defined I/O Attachments

There are two standard procedures which can be used to attach any variable to any file-set.

**Input:** The procedure called INPUT attaches any variable to any file-set such that a reference to the variable reads lines of any specified length from the file-set. The general form is:

```plaintext
INPUT(vname,.fname,reclen)
```

where `vname` is a string containing the name of a simple variable or a reference to a variable; `fname` is a string containing a legal file-set name (for the particular Operating System [see Section 2.3]); and `reclen` is a non-negative integer specifying the length of the strings which will become the value of the attached variable `vname`. Records on the file-set
which are shorter than \texttt{reclen} will be extended with blank characters;
those which are longer are truncated from the right with the extra
characters being lost.

The standard variable \texttt{INPUT} described above has a pre-defined association
defined internally equivalent to the standard procedure call:

\begin{verbatim}
INPUT('INPUT','INPUT',80)
\end{verbatim}

To change the association so that only 72 columns are read the procedure
\texttt{INPUT} can be defined as

\begin{verbatim}
INPUT('INPUT','INPUT',72)
\end{verbatim}

\textbf{Output}: The standard procedure \texttt{OUTPUT} attaches any variable to any file-
set such that a reference to the variable writes lines, with carriage
control if desired, on that file-set. The general form is:

\begin{verbatim}
OUTPUT(vname,.fname,prefix)
\end{verbatim}

where \texttt{vname} is a \texttt{string} containing the name of a simple variable or a
reference to a variable; \texttt{fname} is a \texttt{string} containing a legal file-set
name (for the particular Operating System [see Section 2.3]); and \texttt{prefix}
is either a one character \texttt{string} or not specified. If specified, it will
be concatenated onto the beginning of each string written on the file-set,
to serve as a carriage control character.*

The standard variable \texttt{OUTPUT} described above has a pre-defined association
equivalent to

\begin{verbatim}
OUTPUT('OUTPUT','OUTPUT', ' ')
\end{verbatim}

To punch Hollerith cards, the following association can be defined

\begin{verbatim}
OUTPUT('PUNCH','PUNCH')
\end{verbatim}

It is up to the programmer to limit his output lines according to the
requirements of the storage medium being used.

* See CAL FORTRAN Guide for carriage control characters for the 6400.
Other Standard Procedures for I/O

A number of standard procedures have been incorporated into SNOBOL4 to facilitate the definition and manipulation of file-sets.

1. DETACH(name) removes any input and output association which the variable name may have.

2. ENDCGROUP(name) writes a mark on the file-set name which can be detected later if the file-set is read by the same or another program. (See Section 2.4.)

3. REWIND(name) positions the file-set name at its beginning. If the last operation on this file-set was a write: i.e., a variable attached in the output sense to this file-set was assigned a value, then a group mark will be written on the file-set before it is rewound.

4. EOI(name) where name is a string representing a legal file-set name (see Section 2.3) succeeds and returns a null value if the file-set name is currently positioned at end-of-information (i.e., no more information has been written on the file-set); otherwise it fails.
1.5 Syntax of SN0B0L4

The notation used below in defining the syntax of SN0B0L4 is defined as follows:

1. A class of elements is represented by a notation variable, consisting of script letters.
2. Literal characters are represented by upper case letters or special symbols.
3. Square brackets [ ] denote an option, which may appear or be omitted.
4. The vertical bar | denotes an alternative.
5. Three dots ... denote optional repetition of the immediately preceding syntactic unit.
6. An underline _ indicates the actual character of the SNOBOL syntax; and by itself represents a blank.
7. Braces { } denote a set of alternatives of which one and only one can appear at a given time.
8. Blank spaces appearing in the syntax definitions are only for legibility; only where _ appears are blanks required.

digit::=0|1|2|3|4|5|6|7|8|9
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
identifier::=letter[letter|digit|...⇒ letter { letter | digit | _ | } * * ]
_: = blank, one or more blank spaces.*
integer::=digit...
real::=integer.[integer]
string::= zero or more characters
sliteral::='string1,'
dliteral::="string2"
literal::=sliteral|dliteral|integer|real
simple variable::=identifier

* In addition, blanks may be added to increase readability on both sides of equal signs, commas, semi-colons, and colons, after opening brackets (parentheses and array brackets) and before closing brackets.

1. excluding '
2. excluding "

In addition, blanks may be added to increase readability on both sides of equal signs, commas, semi-colons, and colons, after opening brackets (parentheses and array brackets) and before closing brackets.

1. excluding '
2. excluding "
procedure-call ::= identifier(parameter-list)
name-function ::= procedure-call
field-designator ::= procedure-call
array-element ::= identifier [subscript-list]
variable ::= simple-variable | array-element | field-designator | name-function
$string-primary
string-primary ::= variable | variable | literal | procedure-call | (string-expression)
factor ::= string-primary | factor ** string-primary
term ::= factor | term [*_|/_] factor
sum ::= [+|-] term | sum [+|-] term
string-expression ::= sum | string-expression_sum
  primary ::= string primary | (expression) | variable | primary
  [_||$_|$]variable
pattern-term ::= string-expression | primary
conjunction ::= pattern-term [pattern-term]...
expression ::= conjunction [pattern-conjunction]...
pattern ::= expression
parameter-list ::= [expression], [expression]... [expression] [ { [expression] } ]
subscript-list ::= string-expression [ , string-expression ]...
search ::= string-primary [pattern]
assignment ::= variable [pattern] = [expression]
statement-proper ::= assignment | search
label-part ::= identifier

designational-expression ::= (identifier) | ($primary) [variable]
statement ::= label-part [statement-proper] [goto-part]
goto-part ::= : designational-expression [:S designational-expression]
            [:F designational-expression [:Fd designational-expression[:F designational-expression [:S designational-expression]]]
program ::= { [statement ; | end of card not followed by continuation card] }..END
2.0 **CDC 6400 Machine Dependencies**

The purpose of this chapter is to outline the steps involved in running a SNOBOL4 program on the Control Data Corporation 6000-series computer which operates under the SCOPE(CALIDOSCOPE) Operating System.

2.1 **Deck Structure**

The deck of a basic SNOBOL4 Job is constructed as follows:

- **Deck**
  - **Job card**
  - `<REQUEST Card(s)>
  - `SNOBOL parameters.
  - `: 7-8-9

- **SNOBOL4 Source cards**
  - 7-8-9
  - `<Data cards>`
  - 7-8-9
  - `<Data cards>`
  - `:`
  - `:`
  - 6-7-8-9

**Remarks**
- Field length should be $\geq 12000$.
- If magnetic tapes are used.
- Load-Execute card for SNOBOL compiler. See below for parameters.
- End-of-record marker denoting end of control cards.
- Data read from file-set INPUT by the SNOBOL program.
- Data read from file-set INPUT by the SNOBOL program.
- End of Information denoting the end of the Job.

2.2 **Control Cards**

The Load-Execute card (see CAL FORTRAN Guide, Chapter 15) for the SNOBOL4 compiler may have zero, one or more of the following parameters specified on it:

$$I = \&name$$

where \&name is the file-set from which the compiler will read its input and to which the variable INPUT will be attached. If not specified, the file-set INPUT is assumed.
L = \textit{fname}

where \textit{fname} is the file-set onto which the SNOBOL4 compiler writes the compilation listing and to which the variable OUTPUT is attached. If L = 0 is specified, the listing will be suppressed. If the parameter is omitted, L = OUTPUT is assumed. If compilation errors are detected when L = 0, the remainder of the listing (beginning immediately after the line containing an error) will be written on the file-set OUTPUT.

*B = \textit{num}

where \textit{num} is an octal number to be used as the size of all I/O buffers allocated by SNOBOL: INPUT, OUTPUT and any other file-sets the program references. To be more efficient, the buffer size should be a multiple of the physical record unit size (100\textsubscript{8} for disc files, 200\textsubscript{8} for magnetic tape files) plus one more. If not specified, *B = 201\textsubscript{8} (=129\textsubscript{10}) is assumed. If a number less than 101\textsubscript{8} (=65\textsubscript{10}) is specified, it is replaced by 201\textsubscript{8}.

*F = \textit{num}

where \textit{num} is an octal number to be used as the maximum value to which SNOBOL will increase the field length. If not specified, XF = 30000\textsubscript{8} (12788\textsubscript{10}) is assumed. Therefore, whenever SNOBOL runs out of space during program execution, it will request an additional 1000\textsubscript{8} (=512\textsubscript{10}) words from the Operating System until the value set by this parameter is reached.

Any other parameters will be ignored.

Example:

SNOBOL,*B=403.,I=TAPE2.

requests a SNOBOL execution which reads input from TAPE2 (presumably requested via the appropriate control card), writes the compilation listing on the file-set OUTPUT, uses a buffer size of 402\textsubscript{8} (=257\textsubscript{10}) words and will expand to a maximum field length of 30000\textsubscript{8} CM words.
2.3 File-set names

A legal file-set name for the SCOPE (CALIDOSCOPE) Operating System is one consisting of from one to seven alphanumeric characters, the first of which must be letter: e.g. TESTAPE, OUTPUT2.

2.4 The end of group mark

A SNOBOL group mark, as created by the standard procedure ENDGROUP, creates a logical end-of-record mark of level 00 on a file. Any such end-of-record on a file, whether created by the ENDGROUP procedure or a 7-8-9 card, causes a failure when the attached variable is referenced.

2.5 Source program

The SNOBOL4 source program consists of a sequence of punched cards, of which only the first 72 columns are treated by the compiler although the full 80-column image is written onto the listing.

2.6 Character set

The entire 63 characters of the CDC 6400 character set (see CAL FORTRAN Guide p. 14-16) are available for use in constructing SNOBOL4 programs provided they appear within quotes. Otherwise, only the subset consisting of those characters defined by the syntax are legal with the following exceptions:

- ' is replaced by # (4-8)
- " by + (11-5-8)
- array brackets: [ (7-8) or (/ and ] (0-2-8) or //
- also | is replaced by v (11-0) or //

2.7 Standard Procedure FREEZE

A standard procedure has been incorporated into the SNOBOL compiler for the CDC 6400 to allow the programmer to preserve a compiled version of a SNOBOL program. The general form of the call is:

FREEZE({name})
When FREEZE is entered, it writes out a copy of the entire field length for the job in the format of an absolute overlay, and then terminates execution. The overlay may then be loaded at a later time via a SCOPE (CALIDOSCOPE) control card of the form \texttt{\textbackslash{name}..}. The SNOBOL program will resume execution at the point where FREEZE was called, which is normally just after initialization of patterns, etc.. FREEZE returns \texttt{\textbackslash{name}} as its value.