SNOBOL4

AT

THE UNIVERSITY OF MINNESOTA

TABLE OF CONTENTS

1. INTRODUCTION
   page 1
2. SNOBOL4 AS A PROGRAMMING LANGUAGE
   2.1 History
   page 2
   2.2 Uses
   page 2
   2.3 Implementations
   page 2
   2.4 SNOBOL4 Semantics
   page 3
   2.5 Program Syntax and Semantics
   page 4
3. SNOBOL4 FACILITIES AT THE UNIVERSITY OF MINNESOTA
   3.1 History
   page 9
   3.2 Current Facilities
   page 9
   3.3 Future Possibilities
   page 10
4. COLORADO SNOBOL VERSION 3.10
   4.1 Implementation Features
   page 11
   4.2 Datatypes
   page 11
   4.3 Primitive Functions and Predicates
   page 12
   4.4 Keywords
   page 13
   4.5 Input/Output
   page 13
5. RUNNING PROGRAMS UNDER COLORADO SNOBOL
   5.1 Program Preparation
   page 16
   5.2 The SNOBOL Command
   page 16
   5.3 Command Sequences
   page 17
   5.4 Listing Controls
   page 18
   5.5 Hints for Effective Usage
   page 18
   5.6 Error Messages
   page 19

Appendix A: Summary of Differences Between Colorado and CAL SNOBOL
   page 20
Appendix B: Example Programs
   page 23
References
   page 29
PREFACE

This document is intended to provide information to assist in using the local implementations of SNOBOL4 in a batch environment at the University of Minnesota. Emphasis is put on one implementation, Colorado SNOBOL 3.10, which is a SNOBOL4 processor essentially identical to standard SNOBOL4.

A second processor, CAL 6000 SNOBOL, is also described in part here. For a complete description of CAL 6000 SNOBOL, see the UCC publication entitled "CAL 6000 SNOBOL at the University of Minnesota"; this publication emphasizes interactive uses.

Andrew B. Mickel
Systems Programmer

University Computer Center
University of Minnesota
Minneapolis, Mn 55455

April, 1976

Peter C. Patton,
Director

Lawrence A. Liddiard,
Associate Director,
Systems Operation
This document is not intended to be a guide to learning the SNOBOL4 programming language. Rather, it represents documentation local to the University of Minnesota and describes all SNOBOL4 facilities, particularly those for batch processing. Interactive facilities are described in another UCC publication entitled "CAL 6000 SNOBOL at the University of Minnesota."

Detailed references are given to relevant SNOBOL4 literature both in the text of this document and in the annotated reference list. In particular, two books are widely identified with the SNOBOL4 language: Reference (1) defines standard SNOBOL4, and Reference (2) is intended for persons learning SNOBOL4 as their first programming language. Some of the other references given provide sources of information for CDC 6000/Cyber 70, 170 series SNOBOL4 implementations.

In this document, a brief overview of SNOBOL4 is presented for the purpose of telling what kind of programming language it is. Finally, the SNOBOL4 processors on the Cyber 74 and 6400 computer systems at UCC are discussed, with access information, hints on usage, and examples.
Chapter 2
SNOBOL4 AS A PROGRAMMING LANGUAGE

2.1 HISTORY

According to Ralph Griswold [4], SNOBOL4 is a "general purpose language stressing non-numerical facilities." Earlier versions of this language, SNOBOL and SNOBOL3, were special purpose string processing languages; the operations and data objects available were geared toward sequences of characters. (SNOBOL is an acronym for String Oriented and Symbolic Language.)

The first SNOBOL language was developed in 1962; this development was motivated by the inability of other programming languages to manipulate symbols in mathematical formulas. (The term "symbol" refers to data that does not necessarily involve numerical quantities; rather character representations.) Later, SNOBOL3 evolved to include built-in and programmer-defined functions. In 1967, SNOBOL4 resulted from refinements made to SNOBOL3 in such areas as better pattern matching operations on strings, better arithmetic, and the inclusion of many other data types besides strings of characters [3].

SNOBOL4 is available on many computer systems and has been well documented. The first edition of The SNOBOL4 Programming Language described SNOBOL4 Version 2; the second edition of this text [1] describes SNOBOL4 Version 3 and is now the description of standard SNOBOL4.

2.2 USES

Since 1967 SNOBOL4 has become a major programming language. It is particularly useful because it proves a convenient means for processing data involving character strings; whether the strings are ordinary text, music, computer programs, or their data. A list of some typical applications follows:

a) Text formatting, searching.
b) Game playing.
c) Question-answer systems.
d) Compilation techniques.
e) Language analysis, concordances.
f) Music analysis.
g) Machine simulation.

2.3 IMPLEMENTATIONS

SNOBOL4 was originally implemented on the IBM 360 series computers at Bell Telephone Laboratories. From the beginning, the issue of portability of the interpreter to other systems was considered to be very important. (Earlier versions of SNOBOL had lacked such foresight.) The entire SNOBOL4 processor, except for input/output and operating system interface routines, is written in terms of assembly language macros. About 130 different macros, each designed to perform a task necessary for SNOBOL4 operations or necessary to aid portability to other machines, form a language of their own: SIL (SNOBOL4 Implementation Language). To enable portability, the basic assumption is made that most large scale scientific machines have a macro assembler. SNOBOL4 has been easily transported to many machines in SIL; such implementations are called macro-implementations [3].
Within a year of the first IBM 360 implementation, SNOBOL4 was implemented on a CDC 6000 series machine; the SIL macros were defined in CDC 6000 assembly language (COMPASS) and an operating system interface was provided. SNOBOL4 was also made operational on the Univac 1100 series and the DEC PDP-10 (now the DECsystem 10) computers. All of these macro-implementations are interpretive in nature.

Other kinds of implementations exist also. For example, at the University of California, Berkeley, an interpreter, optimized for the CDC 6000 series was written in COMPASS: this is not a macro-implementation [9]. Later, a true SNOBOL4 compiler was designed at the Illinois Institute of Technology [11] and called SPITBOL (SPeedy Implementation of SNOBOL4); it has been written for the IBM 360 and Univac 1100 computers. A macro-implementation of SPITBOL has been planned.

2.4 SNOBOL4 SEMANTICS

Computer programming language semantics take many forms. Here, SNOBOL4 will be examined from the point of view of available data objects and operations on the data objects. This should explain what the language is designed to do; the most important part of SNOBOL4 is its string manipulation facilities.

A string is a data object consisting of a sequence of zero or more characters chosen from an alphabet. The important properties of strings are:

- Strings can be of variable lengths.
- Strings represent a linked linear list of characters.
- Each character has a position in a string, a successor, and a predecessor.

In SNOBOL4, strings can also have the property of becoming variables; other string values can be assigned to them.

Simple operations on strings include:

- (a) Creating and destroying strings.
- (b) Concatenation of two strings end-to-end, forming a third string.
- (c) Determining the length (size) of a string.
- (d) Duplicating copies of a string.
- (e) Replacing selected characters with others.
- (f) Indirect referencing or accessing the value of a string (when the string is considered a variable).
- (g) Testing for equality and lexical ordering.

Of more importance, a general class of operations called pattern matching can be performed on strings. Pattern matching takes a subject string and a pattern (somewhat like a variable set of strings) and determines whether the subject is in the set of strings specified by the pattern. Besides the yes or no result from pattern matching, partial matching can be achieved and assignment of substrings to variables can take place.

Patterns, therefore, are another important data object in SNOBOL4. Patterns can be created and destroyed as well as assigned to variables. Complex patterns can be produced by building concatenations and alternatives from other pattern elements. SNOBOL4 not only allows literal strings to be used as patterns themselves but also possesses many predefined patterns and pattern components. For example, one component, called LEN, will match any string of a given length.
Around this nucleus of string manipulation features, SNOBOL4 provides other data objects and operations, the presence of which classifies it as a general purpose language. These are:

(a) Integers and real numbers (arithmetic and comparison operations).
(b) Variable names (create, destroy, access from a value).
(c) SNOBOL4 expressions whose evaluation is deferred (evaluation).
(d) SNOBOL4 compiled code (execution, compilation).
(e) Data structuring such as arrays, tables, and programmer defined structures:

Arrays provide a means of numerically indexing a group of related items which are not necessarily all of the same type of data.

Tables associate a collection of variables with their corresponding values much like indirect referencing. Thus, tables are conceptually like singly dimensioned, string-indexed arrays.

Programmer defined structures provide a means of creating arbitrarily linked nodes with any number of fields. They resemble recursive record structures in the PASCAL programming language.

2.5 PROGRAM SYNTAX AND SEMANTICS

Syntax: explains what constitutes correctly formed SNOBOL4 programs, statements, and elements such as variables and values.

Semantics: concerns sequence control, input/output, subprogram parameter transmission, etc.

Discussed here are the syntax and semantics of SNOBOL4 programs. SNOBOL4 has a very loose syntax in that many correctly formed constructions are often meaningless. This fact causes much difficulty when trying to repair a program which is not working correctly and there are no obvious clues.

Strings are written in SNOBOL4 by enclosing the characters forming the string in quotes. SNOBOL4 provides two such quote characters so that one kind of quote can be part of a string. For example

"beautiful, fantastic, fabulous!"
"why can't you do it?"
"Cyber 74"
" " (string of one blank)
" " (string of zero characters or null string)

are all valid (syntactically correct) SNOBOL4 strings. The same kind of quote character must be used as the delimiters of a string. The following examples are incorrect SNOBOL4 strings:

"no"good"at"all"
"illegal literal"

Variable names in SNOBOL4 must begin with a letter and may be followed by any number of letters, digits, or periods. It is easy to see why variable names (which are simply strings) can be treated as strings throughout SNOBOL4.
Statements in SNOBOL4 are all of the following general form:

label subject pattern = object goto

All fields are optional but the subject field must exist to have a pattern or an object field. Each of the fields is placed on a line so that:

(a) Labels must begin in column 1.
(b) Subjects begin at least one blank space after the label field (in column 2 or beyond).
(c) Patterns begin at least one blank after the subject field.
(d) Object fields are separated by a blank, an equal sign, and a blank, from the pattern or subject fields.
(e) The goto field begins with a colon which separates it from the previous field; object, pattern, or subject.

Labels consist of a letter or digit followed by any sequence of letters, digits, or periods.
Comments in SNOBOL4 programs may be placed on lines beginning with an asterisk (*) in column 1.
Continuations of a SNOBOL4 statement from one line require a plus sign (+) or period (.) in column 1 of the next line.
Multiple statements appearing on one line may be separated by a semicolon (;).
Listing directives may be mixed in with the program and begin with a minus (-) in column 1.
All SNOBOL4 programs are terminated by a special statement which has the label END.

Some examples of typical SNOBOL4 statements are:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>VOWELS = 'AEIOU' (assignment statement with subject and object)</td>
</tr>
<tr>
<td>LOOP CARD = INPUT :F(END) (assignment statement, if statement with label, subject, object, and goto)</td>
<td></td>
</tr>
<tr>
<td>NEXT : (START) (goto statement with goto)</td>
<td></td>
</tr>
<tr>
<td>X Y (empty statement with label)</td>
<td></td>
</tr>
<tr>
<td>CARD VOWEL = '×' (pattern matching and replacement statement with subject, pattern, and object)</td>
<td></td>
</tr>
</tbody>
</table>

Note that a blank is a crucial character in SNOBOL4 statements because it is used to separate fields.

SNOBOL4 statements are executed sequentially one at a time but every statement is potentially a goto statement. During execution of a SNOBOL4 statement, the subject, pattern, object, and goto fields (if they exist) are evaluated in order. Evaluation of any field may result in "failure" meaning that a false value is returned. Success or failure may also result from a pattern match. The goto field may specify an S (for success) or F (for failure) in front of a label to determine where the next statement should be executed.
Examples of goto fields are:

:\( (\text{label}) \) \hspace{1cm} \text{Unconditional transfer to label.}
:\( :S(\text{label}) \) \hspace{1cm} \text{If success, then transfer to label.}
:\( :F(\text{label}) \) \hspace{1cm} \text{If failure, then transfer to label.}
:\( :F(\text{label}_1)S(\text{label}_2) \) \hspace{1cm} \text{These are equivalent forms which specify labels in any event.}

Because goto's are the only sequence control mechanism in SNOBOL4, it is hard to write well structured programs. A typical loop looks like this:

\begin{verbatim}
LOOP
    \text{statements forming the body of the loop}
    \text{ }
    \text{N = GT(N,0) N-1 :S(LOOP)}
\end{verbatim}

SNOBOL4 is known as a typeless language. Variables need not be declared to be of any particular data type in a program and can assume values of several different datatypes during program execution. Automatic type conversions are performed among several datatypes; numeral strings, integers, and real numbers are freely converted, one to another.

Variables in SNOBOL4 always have the null string as their initial value and need not be initialized. Manual initialization to the null string can be performed as in these examples:

\begin{verbatim}
STRING1 = ''
\end{verbatim}

or

\begin{verbatim}
STRING1 =
\end{verbatim}

A few examples of SNOBOL4 statements with operations on strings is now in order. Concatenation, or joining two strings together is designated with a blank (another use of the blank character). For example:

\begin{verbatim}
N = N + 1
X = 'THIS IS PROBLEM '
TITLE = X N
\end{verbatim}

The value of the variable TITLE is now: 'THIS IS PROBLEM 6' if the initial value of N was 5.

Indirect reference is designated by a unary $ operator. It is used to reference the value of a string value. For example:

\begin{verbatim}
HARRY = 'COLD'
COLD = 'DARK'
OUT = $HARRY
\end{verbatim}
assigns 'DARK' to OUT. Similarly if

\begin{verbatim}
DOG = 'GRIZZLY'
$DOG = 'BEAR'
\end{verbatim}
assigns the value 'BEAR' to the 'variable' GRIZZLY. Concatenation and indirect reference of strings may be performed in any SNOBOL4 expression within the subject, pattern, object, or goto fields. Multi-way goto's may be constructed by using concatenation or indirect reference in the goto field.
A simple kind of pattern is a string constant. The subject is searched from left to right for the first occurrence of this string; success or failure is the result. For example:

```
CARD 'A' :S(PRINT)
```

will search the value of CARD for an A and if there is one, the next statement will be executed at the label PRINT. Also:

```
CARD 'A' = '*'
```

replaces the first A in the value of CARD with a *.

Other patterns match sets of strings. The pattern element LEN(n) matches strings at least n characters long. Therefore,

```
LINE LEN(5)
```

will succeed if the value of LINE has at least 5 characters. Also:

```
LINE LEN(10) 'A'
```

will succeed if there is an A after the 10th character of the value of LINE. There are many other pattern primitives. Pattern matching may be performed in different modes as well.

Functions in SN0BOL4 may be defined, redefined, and activated recursively. All parameters receive a copy of the value transmitted in the function call. In addition, several variables may be listed in the function definition as being local to the function (their values are protected from those in the main program). The example below defines a recursive function, REVERSE, which returns a string value equal to the argument string with its characters rearranged in reverse order. The variable CH is local to REVERSE.

```
DEFINE('REVERSE(S)CH') :(AROUND)
REVERSE S LEN(1) . CH = :(F(RETURN)
    REVERSE - CH REVERSE(S) :(RETURN)
AROUND
```

Input/output is very simple in SN0BOL4. Any reference to the variable INPUT reads the next line; any new value assigned to the variable OUTPUT gets written on the next line. The two-line program below copies a deck of cards to the printer:

```
NEXT OUTPUT = INPUT :S(NEXT)
END
```
Programs in SNOBOL4 are often structured with variable initializations and function definitions at the top and the rest of the program following. This is illustrated below.

\[
\begin{align*}
\text{var}_1 &= \text{value}_1 \\
\vdots \\
\text{var}_m &= \text{value}_m \\
\text{DEFINE}('\text{fun}_1( )') \\
\vdots \\
\text{DEFINE}('\text{fun}_n( )') : (\text{BEGIN}) \\
\text{fun}_1 \\
\vdots \\
\text{fun}_n \\
\text{BEGIN} \\
\vdots \\
\text{END}
\end{align*}
\]

\{ variable initializations \}
\{ function definitions \}
\{ function bodies \}
\{ main program \}
Chapter 3
SNOBOL4 FACILITIES AT THE UNIVERSITY OF MINNESOTA

3.1 HISTORY

Since 1966, computer users at the University of Minnesota have had access to Control Data 6000/Cyber 70 series computers; a 6600, a 6400, and a Cyber 74. SNOBOL4 was first implemented on a CDC 6600 in 1968 at the Institute for Defense Analyses (IDA) in New Jersey by R. Stockton Gaines [7]. The IDA SNOBOL4 processor was a macro-implementation interpreter similar to the one developed at Bell Laboratories for the IBM 360. This IDA interpreter was improved by Michael Shapiro at Purdue University [8] when Version 2 of SNOBOL4 was released. The Purdue/IDA implementation was made available for classroom instruction at the University of Minnesota in 1969.

Also in 1969, a faster and smaller SNOBOL4 interpreter was written at the University of California, Berkeley that implemented all SNOBOL4 Version 2 features except tracing and dumping [9]. This processor, called CAL-6000 SNOBOL, and referred to here as CAL SNOBOL, was hand coded to run optimally on a CDC 6400. UCC obtained a working copy of CAL SNOBOL in 1970.

New releases of SNOBOL4 at Bell Labs for the IBM 360 were paralleled at Purdue for the CDC 6000. Version 3 of SNOBOL4 was made available in 1971. UCC documented both Purdue/IDA SNOBOL and CAL SNOBOL for batch processing on the 6600 in the summer of 1972.

Because of its small size, CAL SNOBOL gained popularity when it was made available for interactive use on the MERITSS timesharing network in early 1973. A corrected version of CAL SNOBOL (November, 1973) was subsequently overhauled for interactive use for the KRONOS operating system in 1974 by UCC.

An independent effort to implement a macro-implementation for the CDC 6000 was begun by the University of Colorado in 1970 [6]. In time, there were parallel developments of both the Purdue/IDA and Colorado versions of SNOBOL4. With Version 3.9, the Purdue/IDA development was discontinued but the Colorado SNOBOL4 is still active at Version 3.10. Colorado SNOBOL 3.10 is slightly faster and smaller than the Purdue/IDA version; for these reasons, UCC installed Colorado SNOBOL when KRONOS was installed on the Cyber 74 in September 1974. Minor enhancements were made to Colorado SNOBOL by UCC at that time for the KRONOS operating system.

3.2 CURRENT FACILITIES

As of February 1976, the batch SNOBOL4 interpreter on the Cyber 74 has been Colorado SNOBOL Version 3.10 (accessed under the name SNOBOL). This interpreter is recommended for teaching and solving classroom problems because it adheres so closely to the standard SNOBOL4 described in [1], and thus has tracing and other debugging aids.

Also available in batch mode on the Cyber 74 is the CAL SNOBOL interpreter, November 1973 version (accessed under the name SNOBOLC). UCC recommends CAL SNOBOL for serious programming in SNOBOL4 because of its speed (three times faster than Colorado SNOBOL).
The CAL SNOBOL processor is also available in an interactive mode on both the Cyber 74 and the 6400 computer systems. A timesharing subsystem (named SNOBOL) provides easy interactive access. The interactive facilities of CAL SNOBOL are described fully in the UCC publication "CAL-6000 SNOBOL at the University of Minnesota."

3.3 FUTURE POSSIBILITIES

A true SNOBOL4 compiler (such as SPITBOL) with the resultant improvement in execution speed from compiled code would give SNOBOL4 the status it deserves as a really practical language. When a SNOBOL4 compiler for the CDC 6000/Cyber 70,170 series machines is developed, UCC will implement it when practicable. Read the UCC newsletter for new developments concerning SNOBOL4.
Chapter 4
COLORADO SNOBOL Version 3.10

4.1 IMPLEMENTATION FEATURES

Colorado SNOBOL is a macro-implementation interpreter which adheres closely to standard SNOBOL4. As implemented for CDC 6000/Cyber 70,170 series machines, Colorado SNOBOL dynamically requests memory as needed. It is faster than many other macro-implementations because the FORTRAN I/O package is replaced by a streamlined version of its own. At UCC, Colorado SNOBOL does not possess the external function feature which allows linkage to subprograms in other languages.

Like many other SNOBOL4 interpreters, Colorado SNOBOL first compiles the SNOBOL4 program into an internal micro-code and then interprets that. Colorado SNOBOL is in total control of execution and is able to recover from all errors in the user program except time limits.

The remainder of this section will describe the differences between Colorado SNOBOL and standard SNOBOL4 (as detailed in the standard reference The SNOBOL4 Programming Language, by Griswold, et alia.). Where appropriate, pointers (of the form Gm.n) are provided for comparison with the standard reference. (The "G" stands for "Griswold").

4.2 DATATYPES (G1.2, G1.3, G3.6, GA.5, G7.4)

(a) All datatypes except external functions exist.
(b) Integers may range in absolute value up to $2^{34}-1$ (about 10 decimal digits).
(c) Real numbers have a 12-bit exponent and 34-bit (not 48!) mantissa, meaning that about 10 significant digits are preserved with magnitudes ranging up to $10^{\pm 300}$.
(d) Conversion of a string to an integer is limited to 10 significant digits. Conversion to a real number considers the first 10 significant digits (scaling is performed when more digits are encountered). Conversion from a real number to a string generates 6 significant digits. Space has been reserved for a total of 300 characters to allow for scaling because of large exponents.
(e) Strings are chosen from the alphabet of printable characters in the CDC 6000/Cyber 70/Cyber 170 series character set. The collating sequence used is the same as display code:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+-*/()[ ]:\?^\_<>
```


The following table gives the valid substitutions for characters in the standard SNOBOL4 alphabet which are not in Colorado SNOBOL:

<table>
<thead>
<tr>
<th>Standard SNOBOL4 Version 3 (IBM 360)</th>
<th>Colorado SNOBOL Version 3.10 (CDC 6000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>single quote</td>
<td>′</td>
</tr>
<tr>
<td>double quote</td>
<td>″</td>
</tr>
<tr>
<td>less than</td>
<td>&lt; or [</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt; or ]</td>
</tr>
<tr>
<td>vertical bar</td>
<td>V</td>
</tr>
<tr>
<td>ampersand</td>
<td>∧</td>
</tr>
<tr>
<td>at sign</td>
<td>↓</td>
</tr>
<tr>
<td>question mark</td>
<td>↘</td>
</tr>
<tr>
<td>logical not</td>
<td>⊥</td>
</tr>
<tr>
<td>exclamation</td>
<td>™</td>
</tr>
<tr>
<td>percent</td>
<td>%</td>
</tr>
<tr>
<td>number sign</td>
<td>#</td>
</tr>
<tr>
<td>underline</td>
<td>none</td>
</tr>
</tbody>
</table>

4.3 PRIMITIVE FUNCTIONS AND PREDICATES (G3.4, G3.6)

(a) All standard SNOBOL4 functions except BACKSPACE, LOAD, and UNLOAD are available.

(b) The function REPLACE fails if the length of the second and third arguments exceeds 150 characters.

(c) The predicate LGT uses the same lexical ordering as given for the CDC 6000 series display code. Use the value of the keyword ALPHABET as a reference.

(d) Four non-standard primitive functions exist:

1. CLOCK() is a function of no arguments (any given are ignored). It returns a string in the form ^HH:MM:SS^ representing the time of day as given by the operating system. For example, 16:45:00 would be the value of CLOCK() at 4:45 PM.

2. DAYFILE(string) is a function which writes the value of the argument string into the dayfile (operating system log) of the job. DAYFILE fails if SIZE(string) is greater than 80. If SIZE(string) is greater than 40, strings written to the dayfile are formatted into two lines with the excess beyond 40 going to the second line. For example, the statement: DAYFILE(♦**END, PHASE1♦)
will produce an informative message for a program which may have several distinct phases; it may be desirable to know how far execution has progressed. The relevant section of the dayfile for such a job may look like this:

```
17.46.03.SNOBOL.
17.46.05.*END PHASE 1
17.46.05.END -- SNOBOL4
```

(3) LIMIT(unit,count) is a function of two arguments which sets the line limit on an output-associated file specified by the value of the argument unit to the value of count. Line limits on output files are initially set to 5000. The internal SNOBOL4 output routine decrements the limit by 1 each time a line is written onto that unit. The job is terminated when any limit reaches zero. LIMIT returns count as its value unless count is negative, in which case the existing limit for that unit is returned and the limit is unchanged. For example:

```
LIMIT(6,1000)
```

would change the line limit on unit 6 (normally file OUTPUT) to 1000 lines (about 16 pages).

(4) REVERSE(string) is a function of one argument which returns a string whose value is the characters in string in reverse order. For example:

```
OUTPUT = REVERSE(←STINKER←)
```

writes the characters: REKNITS.

4.4 KEYWORDS (G6.1)

(a) ^ABEND, if non-zero at termination of the program, forces an abnormal termination. If a machine dump is requested, look for the value of ^ABEND (modulo 2**18) in register 87 of the exchange package. In all other cases (^ABEND = 0 which is default) SNOBOL, unlike most other KRONOS commands, terminates normally even when errors occur. This means that a SNOBOL job is normally not aborted if compilation errors are encountered.

(b) ^ALPHABET has, as its value, a string of the 63 characters available on CDC 6000 series machines in display code order, followed by a binary zero character (display code 00). This binary zero may be used internally in strings within a program but should not be in strings which are to be output.

4.5 INPUT/OUTPUT (G9.5 - G9.7)

(a) There are 4 I/O units available in Colorado SNOBOL:

<table>
<thead>
<tr>
<th>unit number</th>
<th>external file name</th>
<th>SNOBOL predefined name association</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>INPUT</td>
<td>INPUT</td>
</tr>
<tr>
<td>6</td>
<td>OUTPUT</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>7</td>
<td>PUNCH</td>
<td>PUNCH</td>
</tr>
<tr>
<td>8</td>
<td>SCRATCH</td>
<td>-</td>
</tr>
</tbody>
</table>
Units 5, 6, and 7 are identical to the standard SNOBOL4 implementation. An additional unit, 8, has been added for flexibility. Colorado SNOBOL is limited, however, to a maximum of 4 I/O units because the external function facility (via LOAD) is not available.

The external file names may be changed on the SNOBOL control card; see Chapter 5.

(2) The functions DETACH, ENDFILE, INPUT, and REWIND work exactly as in standard SNOBOL4.

(3) The function BACKSPACE is not implemented.

(4) The function OUTPUT requires further explanation because Colorado SNOBOL uses its own I/O routines instead of the FORTRAN I/O routines. The third argument to OUTPUT is a string which specifies a "FORTRAN format." The useful subset of all possible format specifications are made available in Colorado SNOBOL. Here is a list of allowable format fields:

<table>
<thead>
<tr>
<th>field (general form)</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nAI</td>
<td>Print the next n characters of a string.</td>
</tr>
<tr>
<td>nH...</td>
<td>Print the next n Hollerith characters.</td>
</tr>
<tr>
<td><em>...</em></td>
<td>Print the Hollerith string between the asterisks.</td>
</tr>
<tr>
<td>nX</td>
<td>Insert n blanks.</td>
</tr>
<tr>
<td>/</td>
<td>End a line of printing.</td>
</tr>
</tbody>
</table>

For all format strings:
(a) Nested parentheses are not allowed.
(b) All integers and reals are automatically converted to strings upon printing.
(c) Commas may be used to separate fields but they are not necessary.
(d) Extra spaces are skipped.
(e) The opening parenthesis is also optional.
(f) If n = 1, it may be omitted.
(g) An error found in an OUTPUT format is fatal.

(5) The predefined I/O associations (meaning the variable names attached to unit numbers and hence external file names) are:

INPUT(\INPUT+,5,80)
OUTPUT(\OUTPUT+,6,+(1X,135A1)+)
OUTPUT(\PUNCH+,7,+(80A1)+)

Use of the functions INPUT and OUTPUT can provide extra useful associations; some examples are:
(a) Suppose you have a data file on disk instead of cards. One means of providing access to this data from a SNOBOL program would be to make an association to the spare unit, number 8, as follows:

INPUT(\DATAIN+,8,80)

Then, every time the variable DATAIN is evaluated, a line will be read from the disk file. Note that for unit 8, the disk file name is SCRATCH unless otherwise specified on the SNOBOL control card with the S parameter (see Chapter 5).

(b) Many times it is useful to print listing with carriage controls (page ejects, double spacing, etc.). Output associations can accomplish this. For the variable OUTPUT, only single spacing is provided ordinarily. To create a variable used to print a line at the top of the next page, the following output association may
be used:

\texttt{OUTPUT(NEWPAGE,6,H1)}

Then, NEWPAGE = 'X' would print X at the top of the next page.

(6) Upon reading from any file, execution is terminated if 10 read operations in a row fail. Reading an end-of-record or end-of-file causes a read to fail.

(7) Attempts to write after a read that did not reach an end-of-record are illegal. Files are opened before they are referenced, but never closed.

(8) Use the LIMIT function to specify line limits on output files. See the description of LIMIT earlier in this chapter.
Chapter 5
RUNNING PROGRAMS UNDER COLORADO SNOBOL

5.1 PROGRAM PREPARATION (GA.5)

Colorado SNOBOL is only meant to run in a Batch environment at UCC. This chapter explains the necessary punch card information for setting up Colorado SNOBOL jobs. Previously, we explained the problems with the several special characters in standard SNOBOL4 that are not available on CDC machines. To complicate the problem, keypunch machines at UCC come in two varieties; IBM 026 (no special characters; all must be multi-punched manually) and IBM 029 (converted to the CDC character set); the converted 029 keypunches are recommended. The following table shows the complete character set and the appropriate keypunch codes:

<table>
<thead>
<tr>
<th>Standard SNOBOL4</th>
<th>Colorado SNOBOL</th>
<th>Keypunch (026)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - Z</td>
<td>A - Z</td>
<td>A - Z</td>
</tr>
<tr>
<td>a - z</td>
<td>not available</td>
<td>not available</td>
</tr>
<tr>
<td>_ (underline)</td>
<td>not available</td>
<td>space bar</td>
</tr>
<tr>
<td>(blank)</td>
<td>(blank)</td>
<td>( )</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>: 2-8</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>12-7-8</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>11-5-8</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>' or 4-8</td>
</tr>
<tr>
<td>&lt;</td>
<td>[ or &lt;</td>
<td>7-8</td>
</tr>
<tr>
<td>&gt;</td>
<td>] or &gt;</td>
<td>0-2-8</td>
</tr>
<tr>
<td>&amp;</td>
<td>^</td>
<td>0-7-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>?</td>
<td>+</td>
<td>0-5-8</td>
</tr>
<tr>
<td>@</td>
<td>+</td>
<td>11-6-8</td>
</tr>
<tr>
<td>¬</td>
<td>¬</td>
<td>12-6-8</td>
</tr>
<tr>
<td>#</td>
<td>&lt;</td>
<td>5-8</td>
</tr>
<tr>
<td>%</td>
<td>\</td>
<td>12-5-8</td>
</tr>
<tr>
<td>‰</td>
<td>=</td>
<td>0-6-8</td>
</tr>
</tbody>
</table>
5.2 THE SNOBOL COMMAND (G9.4, G10.1)

Colorado SNOBOL is accessed at UCC by a KRONOS control card named "SNOBOL." The SNOBOL command allows the user to specify the external files which contain:

(a) The SNOBOL4 program and data (unit 5).
(b) The program listing and results (unit 6).
(c) The unit 7 (punch) file.
(d) The unit 8 (scratch) file.

SNOBOL may be executed simply by issuing the SNOBOL command. This is equivalent to issuing another form of the command:

SNOBOL(I=INPUT,L=OUTPUT,P=PUNCH,S=SCRATCH)

Therefore, in general, the SNOBOL command looks like this:

SNOBOL(P1,P2,P3,P4)

where Pn are parameters specifying external file names for the SNOBOL4 I/O unit numbers.

(a) P1 is of the form I=Ifn and is used to specify the logical file name (Ifn) for unit 5 (SNOBOL4 program and data). I=INPUT is assumed; INPUT being the name of the logical file containing information from the card reader in the batch environment.

(b) P2 is of the form L=Ifn and is used to specify the lfn for unit 6 (the program listing and results). L=OUTPUT is assumed; OUTPUT being the name of the logical file normally assigned to the line printer in a batch environment.

(c) P3 is of the form P=Ifn and is used to specify the lfn for unit 7. P=PUNCH is assumed; PUNCH being the name of the logical file normally assigned to the card punch.

(d) P4 is of the form S=Ifn and is used to specify the lfn for unit 8. S=SCRATCH is assumed; SCRATCH being an arbitrary name assigned to a local disk file.

If, for example, you wanted SNOBOL to read your program and data from a disk file named LARRY, and that other data be read in from a file named SALLY (in conjunction with the example I/O association INPUT (†DATA†,8,80) given in Chapter 4), the SNOBOL command may look like this:

SNOBOL(S=SALLY,I=LARRY)

Any or all of the parameters may appear in any order but each must occur only once. If a parameter appears more than once, or if an illegal parameter appears, SNOBOL is aborted. Do not assign the same lfn to two or more parameters because then I/O can get out of sequence.
5.3 COMMAND SEQUENCES

The following is a typical deck structure for Colorado SNOBOL:

Job card with at least CM55000.
USER card.
SNOBOL. (or SNOBOL(parameters))
$ (7-8-9) 
: (SNOBOL4 program and data; only the first 72 columns used)
$ (6-7-8-9)

Note that there is no 7-8-9 card between the program and data; SNOBOL stops compiling upon encountering an END card or when there are two consecutive read failures.

For CAL 6000 SNOBOL, the deck structure may look like this:

Job card with at least CM20000.
USER card
SNOBOLC. (or SNOBOLC(parameters))
$ (7-8-9) 
: (SNOBOL4 program, only the first 72 columns are used)
$ (7-8-9) 
: (data cards, if any)
$ (6-7-8-9)

For explanations of valid SNOBOLC parameters, see the UCC writeup: "CAL 6000 SNOBOL At the University of Minnesota."

5.4 LISTING CONTROLS (G10.1)

Directives to SNOBOL may appear in the SNOBOL4 program. You may find -LIST LEFT and -UNLIST helpful. -LIST LEFT moves statement numbers to the left side of the listing; this is sometimes more convenient. -UNLIST turns off the program listing at that point; this saves paper if the listing is being printed.

Even though SNOBOL lists the first 80 columns of a program, only the first 72 are actually used. Information beyond column 80 is lost, meaning that sequence numbers from the program library maintenance editor, MODIFY, are lost.
5.5 HINTS FOR EFFECTIVE USAGE (G8.1, G11.1 - G11.6)

For most small SNOBOL4 programs, a time limit of 4 seconds is usually sufficient for Colorado SNOBOL on the Cyber 74. The minimum memory space recommended is 55000g but more may be required if you use large amounts of string storage, for example. SNOBOL will dynamically reduce and request memory up to the limit specified on the job card. SNOBOL will attempt storage regenerations (garbage collections) which are time consuming if space is needed; specifying more memory will alleviate this.

Other hints:

1. Define patterns and functions initially, before they are used. A pattern matching statement must build a pattern and then match otherwise.
2. Use ATRIM on input data, whenever possible.
3. If possible, consider efficient pattern matching strategies:
   a. Use anchored mode or POS(0) or FENCE as lead components.
   b. Use BREAK instead of ARB; SPAN instead of ARBNO.
   c. Use ANY instead of alternation on single characters.
   d. Use conditional rather than immediate value assignment.
   e. Use quickscan mode.
   f. Use indirect referencing instead of tables if no prefixing is required.
4. If debugging a program, use at least the following trace and dump features. Insert these 4 statements at the beginning of your program.
   a. ^DUMP = 1 (include for a symbolic (not core) dump)
   b. ^TRACE = 100 (for 100 lines of trace output)
   c. ^ERRLIMIT = 10 (display the first 10 conditionally fatal errors)
   d. ^TRACE(^ERRTYPE,^KEYWORD)

With the ^TRACE = 100 statement, other calls to the TRACE function can be made to print the values of variables; for example, every time their values change. Another example: the statement TRACE(^LINE) would start value tracing for the variable LINE, if it appeared in your program.

5.6 ERROR MESSAGES (G10.1.6, G10.2.3)

Three kinds of error messages come from Colorado SNOBOL:

1. Those relating to an improper call to SNOBOL, involving errors on the SNOBOL command.
2. Compile-time errors; there are 11 of these and they are explained very well in [1].
3. Run-time errors (error terminations); there are 28 of these and the probable causes are explained in [1].
Appendix A

SUMMARY OF DIFFERENCES BETWEEN COLORADO AND CAL SNOBOL

Many times, serious SNOBOL4 programmers at UCC wish to take SNOBOL4 programs that have been developed and debugged under standard SNOBOL4 (Colorado SNOBOL) and then use CAL SNOBOL for production processing to enjoy the improved execution speed. To aid in this conversion, a summary of the differences between Colorado and CAL SNOBOL is given here. Please see the UCC publication "CAL 6000 SNOBOL at the University of Minnesota" for the detailed explanations of the non-standard facilities referred to here.

1. Implementation differences
   (a) CAL SNOBOL is approximately three times faster and three times smaller than Colorado SNOBOL.
   (b) The CAL SNOBOL control card is different: SNOBOLC.
       means SNOBOLC(i=INPUT,L=OUTPUT,D=INPUT,Q=OUTPUT,other options)

2. Syntax
   LABELs may not begin with digits in CAL SNOBOL.

3. Datatypes
   (a) Tables are not implemented.
   (b) Expressions are not a datatype.
   (c) Real numbers have full CDC 6000/Cyber 70,170 series precision: 14 significant digits (48 bit mantissa)
   (d) Integers have 10 digits maximum magnitude.

4. Operators
   (a) Cursor position (^), interrogation (>), negation (¬), and exponentiation (^ or **) are not implemented.
   (b) The unary (*) operator for deferred evaluation is restricted to simple variables (or functions which return variable names) rather than full expressions.

5. Pattern matching
   (a) SUCCEED is not predefined.
   (b) There is no cursor position operator.
   (c) Deferred evaluation is restricted.
   (d) No fullscan mode.

6. Arithmetic
   (a) Mixing integers and reals in arithmetic expressions is not freely allowed. Use the CONVERT function.
   (b) No exponentiation of integers or reals.

7. Primitive functions
   CAL SNOBOL lacks many standard SNOBOL4 primitive functions but possesses some additional ones. Some of these are available in SNOLIB, a library of SNOBOL4 functions in source form for CAL SNOBOL; these are explained in "CAL 6000 SNOBOL at the University of Minnesota." The following list
indicates the status of these functions:

<table>
<thead>
<tr>
<th>Colorado SNOBOL</th>
<th>CAL SNOBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG</td>
<td>use PROTOTYPE</td>
</tr>
<tr>
<td>CLEAR</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>CLOCK</td>
<td>CLOCK</td>
</tr>
<tr>
<td>COLLECT</td>
<td>-</td>
</tr>
<tr>
<td>COPY</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>DAYFILE</td>
<td>REMARK</td>
</tr>
<tr>
<td>DUMP</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>DUPL</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>EVAL</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>FIELD</td>
<td>use PROTOTYPE</td>
</tr>
<tr>
<td>INTEGER</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>LIMIT</td>
<td>-</td>
</tr>
<tr>
<td>LOCAL</td>
<td>use PROTOTYPE</td>
</tr>
<tr>
<td>OPSYN</td>
<td>-</td>
</tr>
<tr>
<td>REMDR</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>REPLACE</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>REVERSE</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>STOPTR</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>TABLE</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>TRACE</td>
<td>in SNOLIB</td>
</tr>
<tr>
<td>VALUE</td>
<td>-</td>
</tr>
</tbody>
</table>

(NOTE: PROTOTYPE in CAL SNOBOL has expanded capabilities.)

SNOBOL4 functions with different actions in CAL SNOBOL:

- ARRAY - no initialization facility.
- CONVERT - only used for numeric datatype conversion.
- ITEM - only on arrays, not tables.
- PROTOTYPE - generalized.

8. Keywords

Some of the standard SNOBOL4 keywords are implemented as primitive functions in CAL SNOBOL.

<table>
<thead>
<tr>
<th>Standard SNOBOL</th>
<th>CAL SNOBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>^ALPHABET</td>
<td>ALPHABET( )</td>
</tr>
<tr>
<td>^ANCHOR</td>
<td>ANCHOR(X)</td>
</tr>
<tr>
<td>^FNCLEVEL</td>
<td>FNCLEVEL( )</td>
</tr>
<tr>
<td>^MAXLENGTH</td>
<td>MAXLENGTH(I)*</td>
</tr>
<tr>
<td>^STCOUNT</td>
<td>STCOUNT( )</td>
</tr>
<tr>
<td>^STLIMIT</td>
<td>STLIMIT(I)*</td>
</tr>
</tbody>
</table>

*The default initial values for these functions are much greater than in standard SNOBOL4.
For the common keywords:

- ATTRIM - use the TRIM function.
- ANCHOR - use ANCHOR(non-null value) to turn on, for example
  \[ \text{ANCHOR(1)} \]
  and ANCHOR(null value) to turn off, for example
  \[ \text{ANCHOR( )} \].

There are no other unprotected and no protected keywords (functions) in CAL SNOBOL.

9. Input/Output
   (a) I/O unit numbers are replaced by sensible KRONOS file names in CAL SNOBOL.
   (b) Only memory restricts the number of files in CAL SNOBOL.
   (c) If magnetic tape files are used directly, B=1001 must appear as a parameter on the control card.
   (d) The I/O function, BACKSPACE, is not implemented.
   (e) DETACH is identical to standard SN0B0L4.
   (f) REWIND and ENDFILE require file names as string arguments.
   (g) Additional functions include ENDRECORD, EOR, EOF, and EO1.
   (h) The proper calls for the functions INPUT and OUTPUT are:

   \[
   \text{INPUT}(+\text{variable}+, +\text{filename}+, \text{number of characters wide})
   \]
   (Note: the default for the third argument is 0, not 80!)

   \[
   \text{OUTPUT}(+\text{variable}+, +\text{filename}+, +\text{carriage control character}+)
   \]

   The third argument to OUTPUT is not a FORTRAN-like format; it is a single line printer carriage control character with default value, null. Strings are thus written as line images of any length, prefixed by the carriage control character. For example,

   \[
   \text{OUTPUT}(+\text{NEWPAGE}+, +\text{OUTPUT}+, 1)
   \]
   will serve as a means of page ejecting in CAL SNOBOL. Strings longer than the width of the line printer are not "properly" wrapped around.

   There is no predefined association for PUNCH. An

   \[
   \text{OUTPUT}(+\text{PUNCH}+, +\text{PUNCH}+)
   \]
   accomplishes this.

10. Listing controls
    (a) Statement numbers always appear on the left.
    (b) -LIST means -LIST LEFT.
    (c) An additional directive, -SPACE, exists that causes a blank line to be inserted in the listing.

11. Error messages
    CAL SNOBOL has different compile-time and run-time messages but these should be self-explanatory.
Two example programs follow; these solve the same problem and produce the same results. The first is a Colorado SNOBOL program, followed by a CAL SNOBOL program that shows typical differences in usage of SNOBOL4 features.

The problem is this: read in English language text on cards and produce an unsorted frequency count of each word in the text.

The programs were run on the University Computer Center Cyber 74 computer system under the KRONOS 2.1 operating system. Timings were:

- Colorado SNOBOL: .626 seconds CP time
- CAL SNOBOL: .284 seconds CP time

This is typical of the execution speed difference between the two processors.
PRODUCE WORD FREQUENCY COUNT

75/06/22. A. MICKEL

THIS EXAMPLE SNOBOL PROGRAM IS RUN UNDER COLORADO SNOBOL
ILLUSTRATING THE USE OF KEYWORDS, THE OUTPUT FUNCTION, THE USE
OF A TABLE AND A USER DEFINED FUNCTION. THE TABLE IS
CONVERTED TO AN ARRAY.
COMPARE WITH THE CAL-6000 SNOBOL EXAMPLE.

THE PROGRAM FLOW:

1. REPEAT
   1. READ IN LINE OF TEXT.
   2. WHILE LINE STILL HAS WORDS IN IT DO:
      1. FIND AND REMOVE NEXT WORD FROM LINE.
      2. STORE IN TABLE AND UP COUNT BY ONE.
   UNTIL NO MORE INPUT
   1. CONVERT TABLE TO ARRAY.
2. PRINT OUT RESULTS.

ATRIM = II DUMP = II ANCHOR = II 1
DUPLHL(N) CREATES STRINGS OF N BLANKS
DEFINE(DUPLHL(N)) i(I(START))
DUPLHL = DUPL(+, +, N) I(RETURN)
START PAT.WORD = SPAN(+,...,II())+ BREAK(+,...,II())+ WORD
OUTPUT(+NEWPAGE+6)+(+1*)*) COUNT = TABLE(20, 20)
NEWPAGE =
OUTPUT =+THE INPUT TEXT IS...
OUTPUT =

READ A LINE OF TEXT
NEXT.LINE LINE = + + INPUT + + IF(OUT)
 LINES = LINES + 1
OUTPUT = LINE

BREAK OFF THE NEXT WORD FROM LINE
LOOP LINE PAT.WORD = IF(NEXT.LINE)
 COUNT<WORD> = COUNT<WORD> + 1 I(LOOP)
OUT OUTPUT = 1 OUTPUT =
OUTPUT = +NUMBER OF LINES PROCESSED = + LINES
OUTPUT =
OUTPUT = +THE FOLLOWING IS A TABLE OF WORD FREQUENCIES IN +
* +THE TEXT ABOVE* OUTPUT = DUPL(+ -62) I OUTPUT =
* COUNT = CONVERT(COUNT + ARRAY+)
I = 1
LUPE OUTPUT = DUPLHL(15, COUNT[1,1]) DUPLHL(30 - SIZE(COUNT[1,1]))
 COUNT[1,2] I(EN),
 I = I + 1 I(LUPE)

NO ERRORS DETECTED IN SOURCE PROGRAM
WHAT THE WORLD NEEDS NOW IS LOVE, SWEET LOVE
IT'S THE ONLY THING THAT THERE'S JUST TOO LITTLE OF!
WHAT THE WORLD NEEDS NOW IS LOVE, SWEET LOVE:
NO, NOT JUST FOR SOME BUT FOR EVERYONE.

(LYRICS BY HAL DAVID)

NUMBER OF LINES PROCESSED = 6

THE FOLLOWING IS A TABLE OF WORD FREQUENCIES IN THE TEXT ABOVE

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHAT</td>
<td>2</td>
</tr>
<tr>
<td>THE</td>
<td>3</td>
</tr>
<tr>
<td>WORLD</td>
<td>2</td>
</tr>
<tr>
<td>NEEDS</td>
<td>2</td>
</tr>
<tr>
<td>NOW</td>
<td>2</td>
</tr>
<tr>
<td>IS</td>
<td>2</td>
</tr>
<tr>
<td>LOVE</td>
<td>4</td>
</tr>
<tr>
<td>SWEET</td>
<td>2</td>
</tr>
<tr>
<td>IT'S</td>
<td>1</td>
</tr>
<tr>
<td>ONLY</td>
<td>1</td>
</tr>
<tr>
<td>THING</td>
<td>1</td>
</tr>
<tr>
<td>THAT</td>
<td>1</td>
</tr>
<tr>
<td>THERE'S</td>
<td>1</td>
</tr>
<tr>
<td>JUST</td>
<td>2</td>
</tr>
<tr>
<td>TOO</td>
<td>1</td>
</tr>
<tr>
<td>LITTLE</td>
<td>1</td>
</tr>
<tr>
<td>OF</td>
<td>1</td>
</tr>
<tr>
<td>NO</td>
<td>1</td>
</tr>
<tr>
<td>NOT</td>
<td>1</td>
</tr>
<tr>
<td>FOR</td>
<td>2</td>
</tr>
<tr>
<td>SOME</td>
<td>1</td>
</tr>
<tr>
<td>HUT</td>
<td>1</td>
</tr>
<tr>
<td>EVERYONE</td>
<td>1</td>
</tr>
<tr>
<td>LYRICS</td>
<td>1</td>
</tr>
<tr>
<td>BY</td>
<td>1</td>
</tr>
<tr>
<td>HAL</td>
<td>1</td>
</tr>
<tr>
<td>DAVID</td>
<td>1</td>
</tr>
</tbody>
</table>
LAST STATEMENT EXECUTED WAS 26

DUMP OF VARIABLES AT TERMINATION

NATURAL VARIABLES

ABORT = PATTERN
1 = PATTERN
BAL = PATTERN
COUNT = ARRAY(27,2)
INPUT = +
ARH = PATTERN
PAT.WORD = PATTERN
LINES = 6
OUTPUT = +
WORD = +DAMID+
LINE = +)
FENCE = PATTERN
FAIL = PATTERN
SUCCEED = PATTERN
REM = PATTERN

UNPROTECTED KEYWORDS

AABEND = 0
ANCHOR = 1
ACODE = 0
AOUTPUT = 1
AEMRLIMIT = 0
ATRAVERSE = 0
AFULLSCAN = 0
AINPUT = 1
AMAXLENGTH = 5000
AOUTPUT = 1
ASTLIDENT = 50000
ATRACE = 0
ATHIM = 1

SN0B0L4 STATISTICS SUMMARY:

176 MS. COMPILATION TIME
388 MS. EXECUTION TIME
234 STATEMENTS EXECUTED, 8 FAILED
100 ARITHMETIC OPERATIONS PERFORMED
46 PATTERN MATCHES PERFORMED
0 REGENERATIONS OF DYNAMIC STORAGE
7 READS PERFORMED
43 WRITES PERFORMED
1.65 MS. AVERAGE PER STATEMENT EXECUTED
This example Snobol program is run under CAL-6000 Snobol illustrating the use of the Anchor function, the user-defined functions DUPL and DUPLb, the Output function, and the use of indirect reference with a protect character (named E) to store the counts of word frequencies. A master reference string is maintained to record each unique instance of a word in the text being processed so that later the frequencies can be retrieved and the results printed.

Compare with the Colorado Snobol example.

ANCHOR(1)
B30 = *
DEFINE(+DUPL(STR*N)DUPLPAT+)
DEFINE(+DUPLb(N)*) I(START)

This is a fast version of DUPL for CAL Snobol
DUPL DUPLPAT = LEN(N * SIZE(STR)) + DUPL
DUPLLOOP STR = STR STR STR STR STR DUPLPAT = (RETURN)F(DUPLLOOP)

DUPLb(N) creates strings of n blanks
DUPLb B30 LEN(N) + DUPLb I(RETURN)

START PAT,WORD = SPAN(+..11..1+) BREAK(+..11..1+) * WORD
E = ++
REFERENCE = + +
NEWPAGE =
OUTPUT = +THE INPUT TEXT IS +
OUTPUT =

HEAD A LINE OF TEXT
NEXT LINE LINE = + + TRIM(INPUT) + + IF(OUT)
LINES = LINES + 1
OUTPUT = LINE

BREAK OFF THE NEXT WORD FROM LINE
LOOP LINE PAT,WORD = IF(NEXT LINE)
REFERENCE = EQ($WRD) REFERENCE WRD + +
$WRD = $WRD + 1 I(LOOP)
OUT OUTPUT = I OUTPUT =
OUTPUT = +NUMBER OF LINES PROCESSED = + LINES
OUTPUT =
OUTPUT = +THE FOLLOWING IS A TABLE OF WORD FREQUENCIES IN +
OUTPUT = DUPL(1++,62) I OUTPUT =

LUPE REFERENCE PAT,WORD = IF(END)
OUTPUT = DUPLb(15) WORD DUPLb(30 - SIZE(WORD)) $(E WORD)
I(LUPE)

SUCCESSFUL COMPIILATION.
WHAT THE WORLD NEEDS NOW IS LOVE, SWEET LOVE
IT'S THE ONLY THING THAT THERE'S JUST TOO LITTLE OF!
WHAT THE WORLD NEEDS NOW IS LOVE, SWEET LOVE!
NO, NOT JUST FOR SOME BUT FOR EVERYONE.

(LYRICS BY HAL DAVID)

NUMBER OF LINES PROCESSED = 6

THE FOLLOWING IS A TABLE OF WORD FREQUENCIES IN THE TEXT ABOVE

<table>
<thead>
<tr>
<th>WORD</th>
<th>FREQUENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHAT</td>
<td>2</td>
</tr>
<tr>
<td>THE</td>
<td>3</td>
</tr>
<tr>
<td>WORLD</td>
<td>2</td>
</tr>
<tr>
<td>NEEDS</td>
<td>2</td>
</tr>
<tr>
<td>NOW</td>
<td>2</td>
</tr>
<tr>
<td>IS</td>
<td>2</td>
</tr>
<tr>
<td>LOVE</td>
<td>4</td>
</tr>
<tr>
<td>SWEET</td>
<td>2</td>
</tr>
<tr>
<td>IT-S</td>
<td>1</td>
</tr>
<tr>
<td>ONLY</td>
<td>1</td>
</tr>
<tr>
<td>THING</td>
<td>1</td>
</tr>
<tr>
<td>THAT</td>
<td>1</td>
</tr>
<tr>
<td>THERE-S</td>
<td>1</td>
</tr>
<tr>
<td>JUST</td>
<td>2</td>
</tr>
<tr>
<td>TOO</td>
<td>1</td>
</tr>
<tr>
<td>LITTLE</td>
<td>1</td>
</tr>
<tr>
<td>OF</td>
<td>1</td>
</tr>
<tr>
<td>NO</td>
<td>1</td>
</tr>
<tr>
<td>NOT</td>
<td>1</td>
</tr>
<tr>
<td>FOR</td>
<td>2</td>
</tr>
<tr>
<td>SOME</td>
<td>1</td>
</tr>
<tr>
<td>BUT</td>
<td>1</td>
</tr>
<tr>
<td>EVERYONE</td>
<td>1</td>
</tr>
<tr>
<td>LYRICS</td>
<td>1</td>
</tr>
<tr>
<td>BY</td>
<td>1</td>
</tr>
<tr>
<td>HAL</td>
<td>1</td>
</tr>
<tr>
<td>DAVID</td>
<td>1</td>
</tr>
</tbody>
</table>

this page has program results
REFERENCES

This annotated list of references for the SNOBOL4 language may be useful for finding information on the language, its implementations, and its applications. References 1 - 5 may be obtained in the University Bookstores or in the Engineering Library. References 6 - 14 are available for reference only in the UCC Reference Room 235a Experimental Engineering.


This standard reference for Version 3 of SNOBOL4 is also an excellent tutorial manual. It describes the macro-implementation of SNOBOL4 implemented on numerous machines. This reference is for persons who have a knowledge of programming in another language.


This is an elementary introduction to computers, programming, and SNOBOL4. It is intended for persons who have no previous technical background or programming experience.


A case study of machine-independent software development. It describes the internal structures and strategies used to implement a SNOBOL4 interpreter in macro assembly language for the IBM 360 and CDC 6000 computers.


This describes a variety of techniques and applications for SNOBOL4. It is an excellent book for the serious, intermediate SNOBOL4 programmer.


This is another excellent book describing applications and techniques in SNOBOL4. Designed for serious, intermediate SNOBOL4 programmers.


A description of Colorado SNOBOL 3.10, the most recent CDC 6000 macro implementation; differences from SNOBOL4 Version 3, implementation information, and external functions facility. Other SNOBOL4 information within this document has been supplied by R. Griswold; it is available from the CDC 6000 User's Group, VIM.


These are the first descriptions of SNOBOL4 for the CDC 6600 as implemented by the Institute for Defense Analyses (IDA). Michael Shapiro of Purdue University based his work on this implementation.

This is the supplementary document for the Purdue/IDA versions of the CDC 6000 macro implementation of SNOBOL4. It describes the differences from SNOBOL4 Version 3 as described in (1); contains implementation information, external functions facility, and bibliography. Available from VIM.


Gaskins, R., **Summary of Functions in CAL SNOBOL4**, October 1969, 17 pages.

Haugeland, W.S., **LO CAL SNOBOL**, April, 1973, 10 pages (VIM report).

All three are supplementary documents for CAL 6000 SNOBOL from the Computer Center, University of California, Berkeley. They explain usage and differences from standard SNOBOL4.


This is an introduction to SNOBOL4 (in particular CAL SNOBOL) which uses numerous examples from the humanities and social sciences.

(11) Easton, J.T., Mickel, A.B., and Strait, J.P., **CAL 6000 SNOBOL at the University of Minnesota**, University Computer Center, 1974, 24 pages.

This document describes CAL SNOBOL with respect to standard SNOBOL4 and also details the University of Minnesota version of CAL SNOBOL; the interactive facilities are emphasized.


Contains a description of a SNOBOL4 compiler for the IBM 360; this compiler is called SPITBOL (SPeedy ImplemenTation of SNOBOL4); discusses extensions and restrictions over standard SNOBOL4.


A series of newsletters informing SNOBOL4 users, implementers, and maintainers of SNOBOL4 developments. Changes to the language, new SNOBOL4 implementations, and SNOBOL4 literature are discussed. Available from Griswold at the University of Arizona.


A series of newsletters promoting SNOBOL4 as a programming language with emphasis on user options. Information is given on various implementations.