930 SNOBOL SYSTEM

REFERENCE MANUAL

Butler W. Lampson

University of California, Berkeley

Document No. 30.50.70

Revised April 18, 1966

Contract No. SD-185

Office of Secretary of Defense

Advanced Research Projects Agency

Washington 25, D. C.

1.0 Introduction 1-1

2.0 Literals 2-1

3.0 Names 3-1

4.0 Expressions 4-1

5.0 Statements 5-1

5.1 Label 5-1

5.2 String Reference 5-1

5.3 Right Half 5-2

5.4 Goto 5-2

6.0 Pattern Matches 6-1

7.0 Functions 7-1

7.1 Built-in Functions 7-2

8.0 Input-Output 8-1

9.0 Executive 9-1

9.1 Teletype Input and Editing Features 9-3

1.0 Introduction

The 930 Snobol system will accept programs written in a language essentially compatible with Bell Labs 7094 Snobol 3. It permits programs to be created, run and debugged interactively.

The basic elements upon which Snobol operates are strings. A string is an arbitrary sequence of 7-bit characters. It is referenced by a string name. Names may appear explicitly in the source program, in which case they must be strings of letters and digits and begin with a letter. Alternatively, they may be constructed by the program during execution, in which case they may be arbitrary strings of characters. Strings may be referenced indirectly; i.e., one string may be used to hold the name of another.

The most important operation in Snobol is the pattern match, in which a reference string may be scanned for the presence of a specified pattern of characters. Particular characters may be scanned for, or a specified number of characters, or a string of characters balanced with respect to parentheses, or a string which matches a string found earlier in the scan. Many combinations of these elements are also possible.

Arithmetic may be done on numeric strings. Built in functions may be invoked to reverse strings, compute string sizes, compare strings for various conditions of equality and inequality and do input-output. The programmer may also define his own functions, which may have local variables and can be recursive to arbitrary depth. All of these facilities are discussed in greater detail in subsequent sections.

2.0 Literals

Literal or constant strings may be introduced into Snobol programs by surrounding them with quotes. Thus

```
'123'
'A REFUSAL TO MOURN THE DEATH, BY FIRE,
 OF A CHILD IN LONDON'
'XT243*X+16.3'
```

are all literals.

Since it is not possible to include the character "single quote" in a literal, the string name QUOTE has as its initial value the string consisting of one single quote. For similar reasons, the strings CR and LF have as their values the strings consisting of one carriage return and one line feed respectively.

To permit the introduction of other non-printing characters, the construct

& octal number

is taken as a literal string consisting of the single character specified by the seven least significant bits of the octal number. Example:

&41 = 'A' &77 = '↑' &147 = bell or G^c

3.0 Names

A name in Snobol may take one of three forms:

a) A simple name, which is a string of letters and digits beginning with a letter. The character '.' (dot) may also appear in names, but should be avoided by the programmer since it is used in various names which are predefined by the system. Example:

A ABC .LE A124

b) An indirection, which is an expression preceded by the character '$' (dollar sign), which may be thought of as a unary operator of high precedence. The argument of the $ is the shortest expression following it. The string value of this expression is obtained, and the name provided by the indirection is exactly this string. Thus

$'ABC'

is exactly equivalent to

ABC

If A = '12'

and B = 'XY'

then $((A+'2') B)

is equivalent to

14XY

c) A function call. Functions are discussed in detail in section 5. The result of writing a function call is that the function is called and execution then proceeds as though the function name alone had appeared. I.e.,

F(X,Y)

is equivalent to

F

except that the function is called with the specified arguments before F is used. Normally, of course, the function will change the value of F.

4.0 Expressions

A Snobol expression is very similar to an algebraic expression, except that the operands of the expression are strings and the value is a string. Among the available operators are several which expect numeric strings as operands, perform some numeric operation, and yield a numeric string as result. A numeric string consists entirely of decimal digits except for the first character, which may be a + or - sign.

Operands may be

a) Names

b) Literals

c) Parenthesized expressions

The operators, in order of increasing precedence, are

+ -	addition and subtraction of integers
* /	multiplication and division of integers
↑	exponentiation
blank	concatenation of strings

unary + -

$	indirection

Examples:

```
A
A+B
A+B C                    ≡  A+(B C)
(A+B) C
A↑'2' + ($B C D)/'16'
F(F(F(A,B),C),F(D,E))
```

The expression

```
('12'*'12') '.'(SQRT ('4')+'16')
```

according to above rules parenthesis must be used.

has the value

```
'144.18'
```

if SQRT is a function defined in the obvious way.

If ABC = 'NEVER'

and XYZ = 'AB'

then $($XYZ 'C') 'UNTIL'

has the value 'NEVER UNTIL'

Note: When used as operators * and / must not be preceded by a blank. When used to signal pattern variables and goto's, respectively, they must be preceded by a blank. Except for this restriction blanks may be used freely. Of course, a blank must not be omitted where a concatenation is intended.

5.0 Statements

Snobol programs are made up of statements, which are ordinarily executed in sequence. Each statement may contain

a label

a string reference

a pattern

a right half

a goto

in that order. Either a string reference or a goto must appear. The following statement contains all of the possible constituents:

```
STAT1  REF   A  *B*   '12'   B   *(C)* =($B+'24'*A))   FCN(C,D)
label  string        pattern                 right half
       reference

                    /S(SUCCESS)F(FAILURE)
                       goto
```

We proceed to define the various constituents of a statement in detail.

5.1 Label

Any statement may have a label. A statement with a label can be referred to in the goto sections of other statements.

The label must be a simple name. It must begin with the first character of the line. A label may be the same as a string name with a value; there will be no conflict, except perhaps in the programmer's mind, between the two uses.

If no label is present, the line should begin with a blank.

5.2 String Reference

The string reference must be the first element of the statement body. If the statement begins with a blank, it is the first object in the statement. Otherwise, it immediately follows the label.

The string reference must be a name or a literal. Its value will be used as the reference string for the pattern match specified by the statement (if any) and will be altered by the right half (if any). If the string reference is a literal there must be no right half.

5.3 Right Half

A right half may appear in any statement which has a name for its string reference. It always consists of a Snobol expression. If the statement does not contain a pattern, the value of this expression becomes the value of the string reference. If there is a pattern, the part of the reference string matched by the pattern is replaced by the value of the right half.

```
ALPHA = BETA
```

is very efficient. Any other kind of right half requires copying of strings.

5.4 Goto

Any statement may have a goto, which must be the last element of the statement. The goto begins with / and may contain any of the following elements:

(name) which causes a transfer to the statement labeled with the specified string name regardless of the success or failure of the statement containing the goto.

S(name) which causes the transfer only if the statement succeeds. This happens if the pattern match (if any) succeeds and all the functions succeed.

F(name) causes transfer to the specified label if the statement fails. This usually happens because of a match failure, but may be due to the failure of any function called in the statement.

The names must all be labels when the goto is actually executed. Both success and failure goto's may appear in the same statement.

Note: S,F or (must be the next character after / in a goto. The / must be preceded by a blank. A goto may appear alone.

6.0 Pattern Matches

Any statement may have a pattern. If the pattern is present, it appears immediately after the string reference. If a pattern occurs in a statement, an attempt is made to match it to the reference string. The attempt will succeed if a compact substring of the reference string can be found which agrees exactly with the pattern. If this happens, control goes to the right half of the statement and then to the success goto, if there is one. Otherwise, the right half is not evaluated and control goes to the failure goto.

A pattern is made up of a series of constituents. Each constituent may be

a) An expression, which can be matched by a substring of the reference string which is identical to the value of the expression. Any arithmetic operations, indirections or function calls in the expression are evaluated before the matching is started. If the expression is a simple name, however, its value may be changed by what happens earlier in the match.

b) A variable, written

name (a blank must precede the first *)

which can be matched by any substring of the reference string, including the null string. If there is more than one possible match, the variable will be matched by the shortest possible substring.

c) A specified length variable, written

name/expression

which can be matched by any substring of the reference string whose length is equal to the (numeric) value of the expression.

d) A balanced variable, written

(name)

which can be matched by any non-null substring of the reference string

which is balanced with respect to parentheses. Thus

A (A) (A)BCD(EFGHI))

can be matched by a balanced variable.

))A((A(B)

cannot.

Patterns can be quite complex. A sequence of examples will illustrate some of the possibilites.

Suppose we let

```
TEXT=  'DEEP WITH THE FIRST DEAD LIES LONDONS DAUGHTER,/
       ROBED IN THE LONG FRIENDS,/
       THE GRAINS BEYOND AGE, THE DARK VEINS OF HER MOTHER,/
       SECRET BY THE UNMOURNING WATERS OF THE RIDING THAMES./
       AFTER THE FIRST DEATH, THERE IS NO OTHER./'
```

Then the statement

```
TEXT   *LINE*   '/'
```

will yield

```
LINE = 'DEEP WITH THE FIRST DEAD LIES LONDONS DAUGHTER,'
```

Then

```
LINE   'DAUGHTER'  ='SON'
```

yields

```
LINE=  'DEEP WITH THE FIRST DEAD LIES LONDONS SON,'
```

For more grandiose examples we define some auxiliary strings:

```
ARTICLE=   'THE '
PREPS =    'OF,IN,WITH,BY,'
PPEND =    'ING'
```

Then the statement

```
TEXT   ARTICLE
```

will succeed, matching the third word. The loop

```
    LOOP   TEXT   ARTICLE =          /S(LOOP)
```

will remove all the articles from TEXT, since the goto causes it to be executed as long as a 'THE' is to be found. Note that if the blank were not present in ARTICLE, the THE in THERE would also be found and removed.

The statement

```
    TEXT   PREPS   ARTICLE
```

will fail, since there is no occurrence in TEXT of all the prepositions in PREPS. To pick out all the phrases of the form

```
    preposition   article   word
```

and remove them, we could write

```
            TEMP1=PREPS
    LOOP1   TEMP1  *PREP* ',' =       /F(DONE)
    LOOP2   TEXT  PREP   ' ' ARTICLE *NOUN* ' ' = /S(LOOP2)F(LOOP1)
    DONE    ...
```

Note that we preserve the original values of PREPS.

We can separate the text into lines, which we will call LINE1, LINE2 etc., with the program

```
    LINENO='0'
    TEMP=TEXT
LOOP1 TEMP *LINE* '/'   = /F(DONE)
    LINENO=LINENO+'1'
  $('LINE'  LINENO) = LINE    /(LOOP1)
DONE  ...
```

After the execution of the two preceding programs we would have

```
     LINE2  ='ROBED'

     LINE4  ='SECRET WATERS THAMES'
```

Observe that we failed to remove all of the prepositional phrases in line 4, since the program we wrote assumed that each phrase would have only three words.

We can look for repetition of words with more than four letters and flag the repetitions with (REP) by writing

```
     NEWTEXT=
LOOP   TEXT   *BEG*  ' '   *WORD*  ' ' *SPACE*  ' '  WORD ' ' *END*
                                                        /F(DONE)

     WORD  *JUNK/'5'*        /S(LONGWORD)
*    WORD IS TOO SHORT OR ALREADY FLAGGED
CONTINUE NEWTEXT = NEWTEXT   BEG   ' '   'WORD'

     TEXT = ' '  SPACE  ' '  WORD   ' '  END    /(LOOP)
LONGWORD   WORD   '(REP)'         /S(CONTINUE)
*    WORD IS GOOD

     NEWTEXT = NEWTEXT  BEG

     TEXT = ' '  WORD ' '  SPACE   ' (REP)' WORD  ' '  END   /(LOOP)
DONE  TEXT=NEWTEXT TEXT
```

If we do this and then the line program, we will have

```
     LINE5 = 'AFTER THE (REP)FIRST DEATH THERE IS NO OTHER.'
```

There will be no other changes.

7.0 Functions

A function may be defined by a line of the form

```
$SCF    FCN(ARG1, ARG2;LVAR1):  FIRSTSTAT
```

This function is called FCN. It has two arguments and one local variable. When it is called the values of ARG1, ARG2 and LVAR1 will be saved. ARG1 and ARG2 will then be given the values supplied by the arguments of the call. LVAR1 will be made null. Control will then go to the statement labeled FIRSTSTAT. A function is called, as described in section 3, by an object of the form

```
FCN( 'ABC', '12' * '24' $XYZ)
```

Each argument may be an arbitrary expression. All the arguments are evaluated just before the function is called.

A goto to RETURN causes an exit from the most recently called function which has not already returned. The old values of ARG1, ARG2 and LVAR1 are restored and the function returns. If the function appears in a context in which its value is significant, the value of the name (FCN) will be the value of the function. Presumably the code for the function will set FCN to something.

Thus, we might write

```
$SCF    REVCAT (A,B) : REV1
REV1    REVCAT = B  A  B    /(RETURN)
```

Then REVCAT ('XY';'Z') has the value

```
'ZXYZ'
```

It is quite legal to write

```
REVCAT ('XY';'Z') = 'ABC'
```

although in this case the function call has no effect except to use up storage.

A function can also return by going to FRETURN. This is said to be a failure exit and causes failure of the statement in which the function occurs, exactly as a pattern match failure does. A function failure in a string reference or pattern prevents any matching and leaves the value of the string reference unchanged. A failure in a right half leaves the string reference unchanged. A failure in a goto is a disaster.

A function must be defined before it is used.

Within each section of the statement (string reference, pattern, right half and goto) all functions are evaluated before anything else is done. The sections of the statement are evaluated in the order indicated.

7.1 Built-in Functions

In addition to Snobol-coded functions which the programmer may define, there is a collection of built-in machine language functions. These are called exactly like Snobol-coded functions. They are:

.ANCH(X) sets the pattern match to anchored mode. X is a dummy.

.UNANCH(X) sets the pattern match to unanchored mode.

.REV(X) reverses the argument string.

.SIZE(X) has the number of characters of X as its value.

.INPUT(X) see section 8

.OUTPUT(X) see section 8

Predicates, which return a null value if they succeed:

.EQUAL(X,Y) succeeds if X and Y are identical strings.

.UNEQL(X,Y) succeeds if X and Y are not identical.

.GRTR(X,Y) succeeds if the string X is greater than Y taken as a string. The initial characters are compared, then the second characters, etc. The longer string is greater if the two strings match throughout the length of the shorter one.

.EQ(X,Y)	succeeds if X and Y are both numeric and X equals Y numerically.
.NE(X,Y)	
.GT(X,Y)	
.GE(X,Y)	
.LT(X,Y)	
.LE(X,Y)	
.NUM(X)	succeeds if X is numeric.
.NULL(X)	succeeds if X is null.

8.0 Input-Output

To input a character, mention the string name CIN. This will cause one character to be read from the current input medium, normally the teletype, and the value of CIN will be set to that character. The character is read when the name is encountered during execution. CIN is not evaluated before evaluation of the string in which it appears like a function call.

To input a line ending with a carriage return, mention LIN. The carriage return character is deleted. Both of these names must appear literally in the source code. If they are obtained by indirection their old values will be provided.

To output a string, set SOUT equal to it. To output a line, set LOUT equal to the line. The carriage return and line feed will be provided. SOUT and LOUT may be obtained by indirection. The output goes to the current output medium, normally the teletype.

To set the input medium, execute INPUT(X), where the value of the argument is the name of the file to be used. Thus, to use the quoted file 'SOURCE', call INPUT('SOURCE'). To set the output medium, use OUTPUT(X).

9.0 Executive

Snobol programs are compiled and run under the control of an executive which accepts a command language similar to that accepted by QED. When the executive is in control, it indicates its readiness for a command by typing

$

It will then accept two kinds of input.

1) Direct statements. Any unlabeled Snobol statement can be typed in. Such a statement should begin with a blank as usual. It will be compiled and if it proves to be correct it will be executed immediately. Control will then return to the executive.

This facility makes it unnecessary to have special commands for examining the values of names or for starting up the program, since both of these functions can be accomplished with suitable direct statements. For example, the statement

```
LOUT = A
```

will print the value of A.

2) Commands, all of which refer to the source language program. A command may have 0, 1 or 2 arguments, separated by commas. Each argument is the address of a line in the program. Such an address consists of a base address preceded or followed by a displacement (or several displacements, if you like). The base address may be

:label:	finds the line with the specified label
[anything]	finds the next line with the specified string in it
.	refers to the current line.

The displacement must be an integer. It may be added by separating it from the base with space or +; or it may be subtracted by using - .

The addressing scheme is identical to that used in QED. The available commands are

PRINT
INSERT
DELETE
CHANGE
EDIT
MODIFY
QUICK
VERBOSE
READ FROM
WRITE ON
/
line feed
↑

which work exactly like the corresponding QED commands;

CODE

which causes the interpretive code generated by each statement to be listed;

NO CODE

which turns off this listing;

BREAK

which inserts a breakpoint at the specified line

KILL BREAK

which removes all the breakpoints in the specified range of statements.

If a statement has a breakpoint, execution of the program will stop each time the statement is reached and the address of the statement will be printed out. The executive will then await commands. The program can be restarted with

GO

All commands except the single character printing ones require a . to confirm their correctness before they are executed.

9.1 Teletype Input and Editing Features

The QED line edit is built into Snobol. Except when the EDIT or MODIFY commands have been used, a statement being typed in is always an edit of the last statement typed. This makes it easy to correct small errors which are detected by the compiler.

When statements are being added with INSERT or CHANGE, they are compiled as they are entered. A statement with a syntactic error will give rise to an error comment and will not be entered.

A statement is always terminated by a carriage return. The line feed character serves as a line continuation character, which starts a new line without terminating the statement. If a line feed is printed out, it has the same effect as a carriage return: to return the carriage and advance the paper.

BAD

We used