AL/COM

PROGRAM: SNOBOL

BY: James R. Guard

Program Bulletin #67-006
Copyright (C)
December 29, 1967
Applied Logic Corporation

AL/COM is a direct access computing service from APPLIED LOGIC CORPORATION

SNOBOL

SNOBOL is a programming language for manipulating strings of characters. SNOBOL'S simple statement formats, simplified Input-Output and automatic storage allocation makes it easy for the novice programmer to learn. On the other hand, the power of SNOBOL's commands for character string manipulation allow elegant and sophisticated programs to be written in SNOBOL. SNOBOL was developed by Farber, Griswold and Polansky of the Bell Telephone Labs in 1962, and was implemented on the IBM 7090. There are SNOBOL languages currently implemented on a number of machines. Write-ups of SNOBOL are contained in SNOBOL, "A String Manipulation Language", Journal of the Association of Computing Machinery, Vol. 11, No. 2 (January, 1964), PP. 21-30, and "The SNOBOL3 Programming Language", The Bell System Technical Journal, Vol. XLV, No. 6, July-August 1966. See "SNOBOL3 Primer", by Allen Forte, 1967, MIT Press for a simple, clear primer in paperback.

While FORTRAN deals mainly with numbers, SNOBOL deals with strings of characters. In FORTRAN a variable usually contains a fixed or floating point number. In SNOBOL, a variable contains a pointer to a string of characters.

The Applied Logic implementation of SNOBOL has a simple means of doing Input/Output using the Teletype and the Disc (or Drum) for reading and writing character strings.

Before going into elaborate detail on the syntax of our implementation of SNOBOL let us examine three simple examples of its use.

EXAMPLE ONE

Example 1 we consider the task of printing a vertical list of words contained in a sentence inputted on the Teletype. Let us assume that a blank, comma-blank, and period are the legal punctuation separating words. Below is a listing of the SNOBOL source for a program implementing this task. The compiling, assembling, and loading needed to run the program are shown. Following the example is a detailed explanation of the action of each line of SNOBOL coding.

EXAMPLE -1-

```
.R PIP2

*TTY:←DRM:EXAMP1
00010   ;EXAMPLE #1
00020   ;INPUT SENTENCE FROM TELETYPE AND MAKE VERTICAL LIST OF WORDS.
00030   ;CONSIDER BLANK, COMMA-BLANK, AND PEROIL AS LEGAL PUNCTUATION.
00040
00050   START:   .NTYPE "*"          ;TYPE * WITHOUT CARRIAGE RET-LINE FEED.
00060            .ACCEPT LINE        ;READ SENTENCE FROM TELETYPE
00070   PARSE:   LINE *WORD* " "!", "!"." =        /F(LAST)
00080            .TYPE WORD          ;TYPE WORD AND CARRIAGE RETURN LINE FEED

00090            /(PARSE)
00100   LAST:    .TYPE LINE          ;TYPE REMAINDER OF LINE
00110                                ;PRESUMABLY EMPTY.
00120            /(START)
00130            .END

*↑C

.R SNOBOL
*EX1.TEM←EXAMP1

NO UNDEFINED SNOBOL ADDRESSES

EXIT
↑C

.R MACRO

*DRM:EXAMP1←DRM:EX1.TEM


THERE ARE NO ERRORS

PROGRAM BREAK IS 000131

5K CORE USED

*↑C

.R LOADER
*SNOOPS,EXAMP1.REL,/G


LOADER
CORE  1

EXIT
↑C
.
```

EXAMPLE -1- (cont'd.)

```
.START

*TODAY IS THE DAY FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY.
TODAY
IS
THE
DAY
FOR
ALL
GOOD
MEN
TO
COME
TO
THE
AID
OF
THEIR
COUNTRY

*
```

Explanation of Example 1

1. A semicolon and characters to the right of a semicolon are considered to be a comment by the SNOBOL compiler and are ignored. Exception: a semicolon between matching quotes is considered as a character in a string. Lines 10, 20, and 30 represent SNOBOL lines which are comments. Lines 50, 60, 80, 100, and 110 show the use of comments at the termination of SNOBOL lines.
2. Blank lines are ignored by the SNOBOL compiler and can be used by the programmer to improve the readability of his coding. For example, see Line 40.
3. START , PARSE , and LAST are words used as address labels in Example 1 in specifying the logical flow of the program. LINE and WORD are words used to name string variables. A label or a string variable is any word made up of letters, numerals, period, percent sign, or dollar sign which does not start with a numeral or a period. A label can be any number of characters, however only the first six characters of a label or a variable are used by the compiler. Words which start with period are reserved for various SNOBOL command functions. A label followed by a colon at the beginning of a SNOBOL line defines the label to be equal to that physical location in the program. For example, see Lines 50, 70, and 100. Such labels serve the same purpose as statement numbers in a FORTRAN program. String variables such as LINE and WORD are defined merely by their appearance. For example, LINE is defined by its appearance in Line 60; WORD is defined by its appearance in Line 70. As in FORTRAN, the appearance of a variable is sufficient to define it. Care must be taken not to use the same word for a string variable and a location label.

4. Blanks and tabs are used as delimiters in a SNOBOL line as needed or desired. For example, in Line 50 no delimiters are needed, while in Line 60 .ACCEPTLINE would be ambiguous without the blank between .ACCEPT and LINE . Note the use of tabs in Lines 50 through 130.

5. The execution of a SNOBOL program starts with the first executable SNOBOL instruction--in this example, Line 50. The label and the command have already been explained. The executable portion of the instruction is the SNOBOL command .NTYPE "*" which requests that the literal string consisting of an asterisk be typed. The second executable SNOBOL instruction requests that a line be inputted from the Teletype and the string of letters corresponding to the line typed be stored and a pointer to this character string be stored in LINE . Though, in fact, LINE contains a pointer to a string of characters, for almost all purposes the programmer can think of line containing a string of characters. In the explanations of the examples we will say that a variable contains a string, when in fact it will contain a pointer to a string. The carriage-return-line-feed which normally terminates a typed line are deleted from the end of the string of characters before being stored. See the sample runs for the asterisk outputted and the sentence inputted by the user.

6. In Line 70 we have a SNOBOL instruction which does not involve input/output and which is more typical of SNOBOL instructions The label PARSE is defined as is described above. The next element of the line is a string variable and indicates that some action is to be taken with the string LINE . The second element *WORD* , is used as a filler in a string matching process which is at the heart of SNOBOL. The third element, " "!" , "?"." represents the

disjunction of three literal strings consisting of a blank. comma-blank, and period. The exclamation point can be read as an "or". A fourth element is the equal sign. This instruction has the following action:

Step A

WORD is set equal to the null (empty) string.

Step B

The first (next) character(s) of LINE is(are) examined to see if it is a blank, comma-blank, or a period. If it is go to Step E.

Step C

The character just examined from LINE is appended to the string in WORD .

Step D

If there is a next character in LINE go to Step B. If there is not, a special dedicated accumulator--called the "test accumulator"--is set to "fail" and terminate the action of the instruction.

Step E

The initial portion of LINE which matches WORD and the blank, or the comma-blank, or the period, is replaced by the string named to the right of the equal sign. In this example there is no string to the right of the equal sign so these initial characters of LINE are simply deleted. The test accumulator is set to "success".

The /F(LAST) which terminates Line 70 indicates that the program is to transfer to the instruction labelled LAST if the test accumulator was set to fail. In the other case, that is, the case where the string match was successful, and hence the test accumulator was set to success, the next instruction (Line 80) is to be executed. Transfer instructions such as /F(LAST) can appear to the right of SNOBOL commands or can stand alone on line as SNOBOL instructions. For example, Line 90 is an unconditional transfer to the instruction labelled PARSE .

7. Line 80 is a command to type the string in WORD on the Teletype. A carriage-return line-feed is appended at the end of each string typed by the .TYPE command.

8. Line 90 is an unconditional transfer back to the instruction labelled PARSE . Looking at the first sample run, at the first time Line 90 is executed, WORD is a five-character string, TODAY and LINE is the string IS THE DAY FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY . In the first sentence tested the loop from instructions 70-90 is executed 16 times and on the 17th execution of Line 70, LINE is the null string and hence the string match requested fails and the program transfers to the instruction labelled LAST .

9. Line 100, which is the instruction labelled LAST , is a request to type the contents of LINE . If a "legal" sentence has been typed at Line 60, LINE will, in fact, be the null line and this accounts for the blank line appearing in our sample run between COUNTRY and the * which indicates to the user that a second sentence is to be typed.

10. Line 120 is an unconditional transfer back to the start of the program. Line 130 contains the SNOBOL command .END This command signals the end of the source to the SNOBOL compiler if this statement is omitted the compiler gives a warning that no .END was found and an .END is assumed.

A detailed description of the SNOBOL language will appear below. Let us comment briefly on the compiling, assembling, loading, and execution of Example 1. The Teletype output from this sequence of operations is shown following the source language for Example 1 above. The .R SNOBOL calls in the SNOBOL compiler from the SYS and begins its execution. It requests a command string from the user by typing an asterisk. No devices should be given in the command string since the DSC (or DRM) is always assumed. The compiler creates a MACRO assembly language program which in our example we have called EX1. TEM . The compiler has told us that here are no undefined SNOBOL addresses, and returns to the monitor. The MACRO assembler is then called in from SYS and the MACRO source file EX1. TEM is assembled. In the example we have called the assembled file EXAMP1.REL . The loader is then called from SYS and our relocatable file, EXAMP1.REL and the SNOBOL operating system, SNOOPS.REL from the SYS are loaded. The SNOBOL operating system is a collection of subroutines which is called by the MACRO coding generated by the SNOBOL compiler from our SNOBOL source coding. Observe that this sample program and the entire SNOBOL operating system load in 1K.

EXAMPLE TWO

As a second example let us consider the SNOBOL program below which accepts a FORTRAN IV source file name from the Teletype, gets in that file from the Disc (or Drum) and replaces the leading spaces from lines in card format,by a tab for Teletype format. The algorithm which we use is as follows:

If a line from the source file does not contain five characters it is assumed to either be a blank line or already be in Teletype format. If a line contains at least five characters and has a tab among the first five characters then the line is considered also to be in Teletype format;

otherwise the line is considered to be in card format. The first five characters of a line are removed and stored in FRONT . All blanks in FRONT are deleted and a tab is appended to the right-hand end of FRONT . These two maneuvers work if there is or is not a statement number. If the sixth character of the line, the continuation character, is blank, it is deleted. Otherwise the sixth character is replaced by the numeral 1. The line is then outputted to the Disc (or Drum) and the next line processed. This program creates a temporary output file for the lines as they are processed. This file is called QQTAB.TEM . After the file is successfully translated the original file is deleted and QQTAB.TEM is renamed to have the name of the original FORTRAN IV source file.

Explanation of Example 2

Lines 10, 30, 40, and 50 are comment lines. Lines 20, 60, and 140 are blank lines. Line 70 types an asterisk without a terminating CR-LF on the Teletype. Line 80 accepts a line of input from the Teletype, strips off the trailing CR-LF and stores the resulting string in FILNAM . Line 90 contains the SNOBOL command to open the input file whose name is stored in FILNAM . If the file named is not on the Disc (or Drum) the SNOBOL operating system (SNOOPS) complains and sets the test accumulator to "fail". The /F(START) terminating line 90 causes a transfer back to Line 70 if the file is not found, otherwise control passes to Line 100.

In Line 100, a Disc (or Drum) file is opened for output with the name QQTAB. TEM .

Line 110 through Line 200 consititute a loop which reads a line of source, processes it; and writes it back out on QQTAB.TEM . In Line 110, the SNOBOL command .READ LINE reads the next line of

EXAMPLE -2-

```
.R PIP2

*TTY:←DRM:EXAMP2
ØØ21Ø   ;EXAMPLE 2
ØØØ2Ø
ØØØ3Ø   ;REMOVE BLANKS, INSERT TABS
ØØØ4Ø   ;DELETE BLANKS IN COLUMN 6 OR REPLACE
ØØØ5Ø   ;NON-BLANKS WITH 1.
ØØØ6Ø
ØØØ7Ø   START:  .NTYPE "*"        ;TYPE * WITHOUT CARRIAGE RET-LINE FEE
ØØØ8Ø           .ACCEPT FILNAM  ;READ FILE NAME
ØØØ9Ø           .OPIN FILNAM    /F(START)
ØØ1ØØ           .OPOUT "QQTAB.TEM"
ØØ11Ø   LINLUP: .READ LINE      /F(EXIT)
ØØ12Ø           LINE *FRONT/5* =        /F(OUTLN2)
ØØ13Ø           FRONT  "        "       /S(TEST2)
ØØ14Ø
ØØ15Ø   BLNK:   FRONT " " =     /S(BLNK)
ØØ16Ø           FRONT = FRONT " "       ;APPEND A TAB
ØØ17Ø   TEST2:  LINE←  " " =    /S(OUTLIN)
ØØ18Ø           LINE *CHAR/1* = "1"
ØØ19Ø   OUTLIN: .WRITE FRONT LINE       /(LINLUP)
ØØ2ØØ   OUTLN2: .WRITE LINE     /(LINLUP)
ØØ21Ø   EXIT:   .RIN ""
ØØ22Ø           .ROUT FILNAM    /(START)
ØØ23Ø           .END

*TTY:←DRM:TEST


        SUM=Ø.
        DO 1Ø I=1,1ØØ
C       FIND THE SUM OF THE NUMBERS FROM 1 TO 1ØØ.
        SUM = SUM + FLOAT(I)
     1Ø CONTINUE
        TYPE 9,SUM
      9 FORMAT(3X,'SUM OF THE NUMBERS FROM 1 TO 1ØØ IS'
       1 F7.2//)
        CALL EXIT
        END

*↑C

.
```

EXAMPLE -2- (cont'd.)

```
 ↑C

.R SNOBOL
*EXAMP2.MAC←EXAMP2

NO UNDEFINED SNOBOL ADDRESSES

EXIT
↑C

.R MACRO

*DRM:EXAMP2.REL,←DRM:EXAMP2.MAC


THERE ARE NO ERRORS

PROGRAM BREAK IS 000225

5K CORE USED

*↑C

 R LOADER
*/SSNOOPS,EXAMP2.REL/D/^


LOADER
CORE  4

EXIT
↑C

.SAVE DRM EXAMP2
JOB SAVED
↑C

.START

*TEST
*↑C

.R PIP1

*TTY:←DRM:TEST
        SUM=0.
        DO 10 I=1,100
C       FIND THE SUM OF THE NUMBERS FROM 1 TO 100.
        SUM=SUM+FLOAT(I)
 1      CONTINUE
        TYPE 9,SUM
9       FORMAT(3X,'SUM OF THE NUMBERS FROM 1 TO 100 IS'
        1 F7.2//)
        CALL EXIT
        END
```

```
*↑C
```

the input from the Disc (or Drum) input file. The trailing CR-LF is deleted, and the resulting string stored in LINE . If no next line exists, the test accumulator is set to "fail" otherwise the test accumulator is set to "success". In Line 110, the transfer to Line 210 is effected when one attempts to read the last-plus-one line.

In Line 120, the first element is the SNOBOL variable Line . This indicates to the compiler that this line is a form of string matching. In this example there is only one element between the first element LINE and the equal sign. This element *FRONT/5* indicates that five characters from LINE should be copied into FRONT . The equal sign has no element to its right, so the first five characters of LINE will be deleted. The effect of Line 120 is that if LINE has five or more characters the first five characters will be removed from LINE and this five-character string is stored in FRONT and the test accumulator is set to success; if LINE has four or fewer characters, then LINE remains the same and the test accumulator is set to "fail." In the case that LINE does not contain at least five characters, transfer passes to Line 200 where the line is written out on QQTAB.TEM and control is then passed to the top of the loop at Line 110. If LINE had five characters control transfers to Line 130.

The effect of Line 130 is that the five-character string in FRONT is searched for a tab. (The string between the two quotes may look like blanks; it is in fact a tab.) Notice in Line 130 that there is no equal sign. Line 130 is an example of string matching whose sole purpose is for program control. In this case, if FRONT contains a tab control is transferred to Line 190. Let us assume that FRONT did not contain a tab. In this case the original line must have been in card format. Hence, control passes to Line 150.

Line 150 is an example of the convenience of SNOBOL. The effect of Line 150 is to remove all blanks from string FRONT (the string between quotes is in fact a single blank). Line 150 works as follows. If a blank is found in FRONT it is replaced by nothing, since nothing appears to the right of the equals. That is to say, the first blank in FRONT if there is one, is deleted. If a blank was in fact deleted the test accumulator is set to success and transfer is passed back to the beginning of the instruction. The transfer is effected by the /S(BLNK) at the end of Line 150. FRONT is then scanned again for a blank. If one is found, it is deleted and transfer is passed again to the front of Line 150. This is continued until all the blanks are removed from FRONT. After a search for a blank which fails, the test accumulator is set to fail and control is transferred to Line 160.

The effect of Line 160 is to append a tab to the right-hand end of FRONT. (The string within the quotes is in fact a single tab.) Line 160 works as follows. The first element in Line 160 is a string variable. This signals the compiler that the line is an application of string matching. Since there is no second element before the equal sign, this indicates that the contents of the first string will be replaced by the contents of the string(s) to the right of the equal sign. In Line 170 we have another application of string matching. Notice however the back arrow immediately following the first element LINE. This indicates that the string being matched must include the first character of LINE. The effect of this line is to delete the leading character of LINE if it is a blank and transfer control to Line 190 or to do nothing to LINE and fall through to Line 180 if the first character of LINE was not a blank. Since this character being examined was originally the sixth character of the source line, this instruction tests the continuation field. In Line 180 we replace the first character in LINE, if there is one, by the numeral 1

If at this point LINE were empty Line 180 would not alter LINE , Control then passes to Line 190.

Line 190 causes the string in FRONT concatenated with the string in LINE to be outputted along with a CR-LF to QQTAB.TEM . Control is then passed to the top of the loop at Line 110. After all lines have been inputted, the read command at Line 110 will fail and control will pass to Line 210. The SNOBOL command in Line 210 calls for the input file to be renamed to a file with a null name. That is, the null string which is generated by two adjacent quotes, represents a null name. The effect of renaming an input file to a null name, is to delete that file. Control is then passed to Line 220.

In Line 220, the SNOBOL command .ROUT FILNAM renames the output file to be the name contained in FILNAM . Control is then passed to Line 70 so that the user can insert a new file name for processing.

EXAMPLE THREE

As a final example, let us consider the SNOBOL program below which inputs a list of words, written one word to the line, and outputs the list as a continuous stream of characters separated by commas, and then outputs an alphabetized list in the same format. The first word of the input list is a number which gives the number of letters in the longest word in the list. This number will be deleted before the list is printed.

Explanation of Example 3

In Lines 30 through 50 the program types an asterisk, accepts a file name typed on the Teletype and opens that file. If the file is not found on the Disc (Drum) the program is restarted. In Line 70 we read the first line of the drum input file. If the file is empty, the test accumulator would be set to "fail" and the transfer at the right-hand end of Line 70 would be effected. The string in SIZE is considered by this program to be a SNOBOL integer. Any SNOBOL string can be considered to

EXAMPLE -3-

```
;ALPHABETIZATION USING A RADIX SORT TECHNIQUE

BEGIN:  .NTYPE  "*"
        .ACCEPT FILNAM
        .OPIN   FILNAM  /F(BEGIN)

;READ MAXIMUM SIZE OF WORD
START:  .READ   SIZE    /F(BEGIN)
        SIZE > "0"      /F(BEGIN)

;READ ALL WORDS - SEPERATE WITH COMMAS.
        LIST =
READER: .READ   WORD    /F(TYPE1)
        LIST = LIST WORD ","    /(READER)

;TYPE LIST TO BE ALPHABETIZED
TYPE1:  .TYPE "LIST TO BE ALPHABETIZED:  " LIST

DECSIZ: SIZE = SIZE - "1"
        SIZE < "0"
        /S(FINAL)

GETWRD: LIST *WORD* "," =       /F(REMAKE)
        WORD    *HEAD/SIZE*     *PIT/1*
        /F(STOBIN)
        @PIT = @PIT   WORD ","  /(GETWRD)
STOBIN: BIN = BIN WORD ","      /(GETWRD)

REMAKE: BIN *LIST* =
        ALPHA = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
NXTLET: ALPHA *PIT/1* =         /F(DECSIZ)
        LIST = LIST @PIT
        @PIT =          /(NXTLET)

FINAL:  .TYPE "ALPHABETZED LIST:  " LIST
        /(BEGIN)
        .END

*
```

```
 ↑C

.R SNOBOL
*EXAMP3.MAC←EXAMP3/1ØØL

NO UNDEFINED SNOBOL ADDRESSES

EXIT
↑C

.R MACRO

*DRM:EXAMP3.REL,←DRM:EXAMP3.MAC


THERE ARE NO ERRORS

PROGRAM BREAK IS ØØ3513

5K CORE USED

*↑C

.R LOADER
*/SSNOOPS,EXAMP3.REL/D/G


LOADER
CORE  4

EXIT
↑C

.SAVE DRM EXAMP3
JOB SAVED
↑C

.R PIP1

*DRM:TEST←TTY:
3
TESTED
TRIED
ALWYAS
GIFT
XMAS
ACCEPT
BECAUSE
HOSPITAL
↑Z
*
```

EXAMPLE -3- (cont'd.)

```
 3 DRM EXAMP3 4
JOB SETUP
↑C

.ST

*TEST
LIST TO BE ALPHABETIZED:  TESTED,TRIED,ALWAYS,GIFT,XMAS,ACCEPT,BECAUSE,H
OSPITAL,
ALPHABETIZED LIST:  ACCEPT,ALWAYS,BECAUSE,GIFT,HOSPITAL,TESTED,TRIED,XMAS
.
*↑C
```

be an integer by reading the initial numerals as a decimal integer. The end of the string or the first non-numerical character ends the number. The only exception to this rule is that a leading minus sign makes the number which follows negative. For example, "-", "0", "", "000" "0.3" "0" "+3" "--5" are all considered to be zero as a SNOBOL integer. Also, "-5", "-5.5", "-5A" are all considered to be -5 . In Line 80 we test to see if SIZE is indeed a positive integer. Notice the use of the literal zero. This statement could also have been written

```
      "1"< = SIZE            /F(BEGIN)
```

The operation of Line 80 is performed as follows. A routine in the SNOBOL operating system (SNOOPS) evaluates SIZE as a SNOBOL integer. Similarly it evaluates a string consisting of a zero alone as a SNOBOL integer and sets the test accumulator to "success", --if the value of SIZE is currently greater than zero.

The transfer at the right hand side of Line 80 is effected if the first characters in SIZE are not considered to be a positive integer. In Lines 110 through 130 we read all the words and make them into a single string separated by commas. In Line 160 we type the list to be alphabetized on the Teletype. In Line 180 we reduce the size of SIZE by 1 and Line 190 we test to see if SIZE is negative. In Line 200 we transfer to the final output coding if SIZE is negative.

The loop from Line 220 through Line 260 is used to sort all the words on LIST into 27 "bins." There is one bin for each letter of the alphabet and the 27th bin for words which are less than or equal to the number currently stored in SIZE . The initial use of this loop sorts all words whose length is less than the maximum length into the bin called BIN , and sorts the longest words into each of the bins A, B, C,...,Z according as its last letter is A, B, C,...,Z respectively. On the second application of this loop, the words of maximum length are sorted on the next-to-the-last letter, the words of one less than maximum length are sorted on their final letter and words shorter than that are sorted into BIN .

In Line 220 the first word in LIST is copied into WORD and the initial word and comma are deleted from LIST . However, if LIST was empty, transfer is made to Line 280. In Line 230, we see two different types of application of fixed length fillers. The first filler.

HEAD/SIZE

is interpreted as follows. SIZE is interpreted as a SNOBOL integer and that number of initial letters of WORD is copied into HEAD . The second filler is a fixed length filler where the size is given by an integer so that the next character after the last character read into HEAD is stored in PIT . If this is successful the test accumulator is set to "success." However, if WORD is too short, i. e., does not contain SIZE+1 letters, the test accumulator is set to "fail." Line 240 effects a transfer to Line 260 in the case that WORD was too short. In Line 250 the list whose name is in PIT is lengthened by adding the contents of WORD and a comma. The commercial-at sign (@) in front of PIT indicates that it is not the contents of PIT but the contents of the string named in PIT that is being referred to this feature is known as indirect naming. If the string in PIT happened itself to be a variable name preceeded by a commercial at-sign the indirect naming would go down one level deeper. Indirect naming can be used to an arbitrary level of indirectness. Indirectness can also be used with addresses. That is, a transfer can be made to a label named in a SNOBOL string. Line 260 is entered in the case that WORD is shorter in length than the number in SIZE . In Lines 280 through 320, LIST is regenerated by putting the shorter words followed by all of the words in List A, List B, . . . , List Z . Line 280 represents an interesting use of a filler. The action here is to put the contents of BIN into LIST and to empty BIN . Notice that fillers are used to match the smallest possible string except in the case that a filler appears at the beginning or end of the substring being matched. A filler at the beginning or end of the substring, if it were to be as short as possible would always be empty. Therefore, it is convenient to have an initial filler to include all of the beginning of the main list and a filler at the end of the substring to include all of the characters at the end of the main list. Hence, a substring consisting only of a filler matches the entire main string. In the case of Line 280, LIST matches all of BIN and the equal sign indicates that the contents of BIN is to be deleted. Lines 300 through 320 are the loop which adds the 26 lists to LIST . After 26 passes through the loop, control is transferred to Line 180, where SIZE is decremented, etc. When SIZE has been decremented to -1, control transfers to Line 340 where the alphabetized list is printed out. After this typeout, control has passed to the beginning of the program for the user to type in a new file name.

We now describe the syntax of the Applied Logic implementation of SNOBOL. In most respects we have followed the syntax of the original implementations of SNOBOL. If we have diverted, it is because we have the ASCII character set available and because the ASSEMBLE/COMPILE feature (i.e., the in-line use of the MACRO assembly language code) makes it desirable to have the syntax for comments and labels be compatible with MACRO assembly language.

Literal Strings

1 "" is a literal string representing the null (or empty) string.

2 """" is a literal string representing a string consisting of a quote alone.

3. "$L_1L_2...L_n$" is a literal string, where $L_1L_2...L_n$ are any ASCII characters other than quotes ($n\geq 1$).

   NOTE: Strings are stored internally as a continuous string of ASCII characters terminated by a null (a null is a 7-bit field of all binary zeros).

Names

Any word using only letters of the alphabet, numerals, percent sign or period--that does not start with a numeral or a period is a name. Names are used as "string names" to name string variables and as address labels. The compiler allows names of any non-zero length. However, only the first six letters are used by the compiler. For example, ABCDEF1 and ABCDEF2 are both legal names which are considered to be the same name by the compiler.

Labels

A name which is initially placed in a line and immediately followed by a colon is a label. Care should be taken not to allow label and string names to coincide.

String Element

1. A string name is a string element.

2. A literal string is a string element.

Substring Element

1. A string element is a substring element.

2. If S and T are string elements but not string literals and if n is a positive decimal numeral, then

   a) *S*

   b) *S/n*

   c) *S/T* and

   d) &S&

   are substring elements.

3. If S_1, S_2,...,S_n ($n\geq 2$) are string elements, $S_1!S_2!...!S_n$ is a substring element. (For example, "VAR"! "12"!VAR3!"345" is a substring element.) The ! is read as OR.

NOTE: Substring elements are concatenated together to make a pattern which is to match a designated character string.

Comments

A semicolon, not captured within part of a literal, together with all characters to the right of such a semicolon, are considered to be a comment.

1. Format: $F\ S_1\ S_2\ \ldots.\ S_n \quad (n \geq 1)$

   Action: Sets test accumulator to "success" if $S_1\ S_2\ \ldots\ S_n$ matches a consecutive substring of F; otherwise sets test accumulator to "fail."

2. Format: $E\ S_1\ S_2\ \ldots.\ S_n = R_1\ R_2\ \ldots.\ R_m \quad (m \geq 0,\ n \geq 0)$

   Action (n=0): Replace E by $R_1\ R_2\ \ldots.\ R_m$. No effect on test accumulator.

   Action (m=0): Delete first substring of E matched by $S_1\ S_2\ \ldots.\ S_n$ and set test accumulator to "success;" if no match, leave E in tact and set test accumulator to "fail."

   Action (n>0, m>0): Replace first substring of E matched by $S_1\ S_2\ \ldots.\ S_n$ with the string $R_1\ R_2\ \ldots.\ R_m$ and set the test accumulator to "success;" if no match, leave E in tact and set test accumulator to "fail."

3. Format: $E\ S_1\ S_2\ \ldots.\ S_n = A \quad (n \geq 0)$

   Action: Same as 2. above with the character string which represents the value of A used in lieu of $R_1\ R_2\ \ldots.\ R_n$.

4. Format: $E\ S_1\ S_2\ \ldots.\ S_i \rbrack\ S_{i+1}\ \ldots.\ S_n = R_1\ R_2\ \ldots.\ R_m$

   $(0<i<n,\ m \geq 0)$

   Action: Same as 2. above except that only the portion of the substring matched by $S_1\ S_2\ \ldots.\ S_i$ is replaced or deleted.

5. Format: $E\ S_1\ S_2\ \ldots.\ S_i \rbrack\ S_{i+1}\ \ldots.\ S_n = A \quad (0<i<n)$

   Action: Analogous to 4. above.

6. Format: $E\ S_1\ S_2\ \ldots.\ S_i \lbrack\ S_{i+1}\ \ldots.\ S_j \rbrack\ S_{j+1}\ \ldots.\ S_n =$

   $R_1\ R_2\ \ldots.\ R_m \quad (0<i<j<n,\ m \geq 0)$

   Action: Analogous to 4. above except that the substring of E corresponding to $S_{i+1}\ \ldots.\ S_j$ is replaced or deleted.

7. Format: $E\ S_1\ S_2\ \ldots\ldots\ S_i \left[S_{i+1}\ \ldots\ldots\ S_j \right] S_{j+1}\ \ldots\ldots\ S_n = A$

$(0<i<j<n)$

Action: Analogous to 6. above.

8. Format: $E\ S_1\ S_2\ \ldots\ldots\ S_j \left[S_{j+1}\ \ldots\ldots\ S_n \right. = R_1\ R_2\ \ldots\ldots\ R_m$

$(0<j<n,\ m\geq 0)$

Action: Analogous to 4. above except that the substring of E corresponding to $S_{j+1}\ \ldots\ldots\ S_n$ is replaced or deleted.

9. Format: $E\ S_1\ S_2\ \ldots\ldots\ S_j \left[S_{j+1}\ \ldots\ldots\ S_n \right. = A$

$(0<j<n,\ m\geq 0)$

Action: Analogous to 8. above.

In ord r to understand the operation of the string command, consider the command

NAME SUB1 SUB2....SUBn = REP1 REP2...REPm

This statement is executed in the following manner:
SUB1, SUB2...SUBn are to indicate a contiguous substring of NAME which is to be replaced by the string obtained by concatenating the strings represented by REP1, REP2...REPm . If this substring match is successful, the test accumulator (T=17) is set to "success" (0); if the match fails, the test accumulato is set to "failure" (-1).

The matching algorithm proceeds as follows:

The substring elements of the form *S* , *S/j* , *S/T* , and &S& represent fillers whose lengths are respectively "the shortest possible," "exactly j(j ≥ 1) characters," "exactly t characters--where T has value t as a SNOBOL integer--," and "the biggest possible." The matching algorithm then attempts to find the left-most match for SUB1 and NAME and then proceeds to find immediately after that a match for SUB2 and so on through SUBm . If for SUBi+1 , a match is not possible, an attempt is made to extend the right-hand end point of SUBi one character to the right--in the case of &S&, contract the right end point one character to the left. If this is not possible then SUBi-1 is considered, etc. If this fails back through SUB1 the left-hand end point of SUB1 is incremented 1 to the right and the process is restarted.

If SUB1 SUB2...SUBn does not appear in the command string, the whole string NAME is considered to be matched. If REP1 REP2 ...REPm is missing the matching substring of NAME is to be deleted. If = REP1 REP2...REPm is missing (then n >0), this operation is used strictly for setting the fail-success accumulator. For example

```
NAME = ",A,B,C,"
NAME "," *VAR* "," =
```

results in NAME being B,C, and VAR being A and the test accumulator is set to "success."

```
NAME = ",A,B,C,"
NAME *DELIM/1* *VAR* DELIM
```

sets DELIM to , and sets VAR to A . Again, the test accumulator is set to "success."

It has been found to be convenient to make replacements within a proper substring of the substring being matched. SNOBOL delimits such a substring by including one or both of [or] between the SUBi's. A missing [is tacitly assumed to appear before SUB1 ; similarly a missing] is assumed to follow SUBn . For example

```
NAME = ",A,B,C,"
NAME *DELIM/1* *VAR* ] DELIM =
```

results in NAME being set to ,B,C, and VAR being A and DELIM being , . Note that the comma preceeding B is not replaced (deleted) since replacement is restricted to the bracketed substrings.

SUBi can, in addition, be a disjunction of string names or literals. For example , replace the right-most occurrence of period or comma by a semi-colon:

```
NAME = ",A,B,C,"
NAME &VAR& [ "."!"," = ";"
```

results in NAME being ,A,B,C, in VAR being ,A,B,C and in the test accumulator being set to "success."

An additional feature of SNOBOL is the anchor mode for matching the substring. In this mode the matching substring must include the first character of the string. This mode is indicated by an ← immediately following the first string name. Spaces and tabs are used in SNOBOL as syntactic delimiters for readability and in a few cases to remove ambiguities caused by names running together. The delimiters however are not needed in general except in cases of such ambiguities.

Arithmetic Relations

Let A and B be string elements. Then

A == B (is A equal to B ?)

A # B (is A unequal to B ?)

A< = B (is A less than or equal to B ?)

A < B (is A less than B ?)

A > B (is A greater than B ?)

A> = B (is A greater than or equal to B ?)

are arithmetic relations. Note that A and B can not themselves be arithmetic terms. An arithmetic relation sets the test accumulator to "success" or "failure" according as the relation holds or does not hold.

Transfer Command

Let L_1 and L_2 be SNOBOL names which are used as labels or are string names preceeded by a commercial-at sign or a dollar sign. Then

$/(L_1)$ (transfer to L_1)

$/S(L_1)$ (transfer to L_1 if test accumulator is "success")

$/F(L_1)$ (transfer to L_1 if test accumulator is "fail")

$/F(L_2)S(L_1)$ (transfer to L_1 if test accumulator is "success", [illegible]

$/S(L_1)F(L_2)$ transfer to L_2 if test accumulator is "failure")

are all transfer commands. If desired, a transfer command may be located to the right of a string command, an arithmetic relation, or a SNOBOL command.

SNOBOL Commands

Let $E_1, \ldots, E_n$ be string elements. Let S be a string name. Let L be a label address or a string name preceeded by a commercial at sign or a dollar sign. The legal SNOBOL commands have the following formats. Use of SNOBOL commands will be described below.

.TYPE	$E_1\ E_2 \ldots E_n$	$(n \geq 1)$
.NTYPE	$E_1\ E_2 \ldots E_n$	$(n \geq 1)$
.ACCEPT	S	
.OPIN	S	
.OPOUT	S	
.APOUT	S	
.WRITE	$E_1\ E_2 \ldots E_n$	$(n \geq 1)$
.NWRITE	$E_1\ E_2 \ldots E_n$	$(n \geq 1)$
.READ	S	
.CLIN		
.CLOUT		
.RIN	S	
.ROUT	S	
.SNIN		
.NSNIN		
.SNOUT		
.NSNOUT		
.PUSHJ	L	
.POPJ		
.POP		
.MACRO		
.SNOBOL		
.EXIT		
.END		

.TYPE $E_1 E_2 \ldots E_n$ ($n \geq 1$)

The strings referenced by the string elements which appear to the right of a .TYPE statement are concatenated together and outputted on the Teletype and a terminating CR-LF is typed. For example

```
        .TYPE   "A"    "B
C"
        .TYPE   "D"
```

outputs

```
AB
C
D
```

on the Teletype. This command does not affect the test accumulator.

.NTYPE $E_1 E_2 \ldots E_n$ ($n \geq 1$)

The .NTYPE statement is similar to the .TYPE statement except that the terminating CR-LF is not affixed. For example

```
        .NTYPE   "A"    "B
C"
        .TYPE   "D"
```

outputs

```
AB
CD
```

on the Teletype. This command does not affect the test accumulator.

.ACCEPT S

This statement readies the Teletype for input, deletes S , reads in the first line of input into S . In both the .ACCEPT statement and the .READ statement described below the input is terminated by a character which represents a vertical movement. These characters are the line-feed, form-feed, vertical tab, and the line-feed generated by a carriage return (recall that the return character generates a CR and a LF). In the case that the input line is terminated by a CR-LF or a LF these terminating characters are deleted from the input line. The vertical tab and the form feed however are stored with the line. It has been found that these conventions for input are very convenient in SNOBOL. This command does not affect the test accumulator.

.OPIN S

This statement opens an input file on the Disc (Drum) whose name is contained in S . For example

```
        .OPIN    "ABC.EXT"
```

looks on the user's directory for the file ABC.EXT . The operating system, SNOOPS , complains if the file is not found or not accessible The test accumulator is set to "success" if the file is found and is accessible. The test accumulator is set to "fail" in the contrary case. At most, one input file can be opened at a time from the Disc (Drum).

.OPOUT S

This statement opens an output file with the name contained in S . At most, one output file on the Disc (Drum) can be opened at any time. This command sets the test accumulator to "success" if the file was successfully opened and to failure in the contrary case.

.APOUT S

This statement is analagous to the .OPOUT statement except that the file named in S should be an existing file and output is to be appended on to the end of that file. SNOOPS complains if the file named in S does not exist.

.WRITE E_1 E_2...E_n ($n \geq 1$)

The .WRITE statement is analagous to the .TYPE statement except that output is to the Disc (Drum) file opened by the last .OPOUT or .APOUT statement. In both the .TYPE and .WRITE statements a CR-LF is affixed to the end of the line if the characters typed do not terminate in a vertical tab or a form-feed. The test accumulator is not affected.

.NWRITE E_1 E_2...E_n ($n \geq 1$)

This statement is identical to the .WRITE statement except that in no case is a CR-LF affixed.

.READ S

This statement is analagous to the .ACCEPT statement except that input is the next line of the Disc (Drum) file opened by the previous .OPIN statement. If no Disc (Drum) input file is open SNOOPS complains. The test accumulator is set to "success" if a line was successfully read. In the contrary case, the test accumulator is set to "fail."

.CLIN

This statement closes the input file.

.CLOUT

This statement closes output file. It is not necessary to execute the .CLIN or .CLOUT statements unless one is going to open a new input, respectively output, file on the Disc (Drum).

.RIN S

This statement closes the input file currently open and renames it to the name in S . If S contains the null string the input file is deleted rather than renamed.

.ROUT S

This statement is alalagous to the .RIN statement except that it renames or deletes the output file opened on the Disc (Drum).

.SNIN

The normal mode of reading input from the Disc (Drum) strips sequence numbers from the lines if they appear. This statement acts as a switch which initiates the feature that sequence numbers are read in as part of the line. The format of the sequence number is as follows. The first five characters are numerals; the sixth character is a tab unless the content of the line is to be empty, in which case the sixth character is a line-feed. Hence the .SNIN feature will cause .READ to read in six characters plus the remaining characters of a line (or read in exactly five numerals, since the terminating line-feed is stripped, in the case that the line represented an empty line with a sequence number). The remaining commands do not affect the test accumulator.

.NSNIN

This statement turns off the feature initiated by the .SNIN.

.SNOUT

The normal mode for output on Disc (Drum) does not generate sequence numbers. This statement initiates the feature that considers the first six characters of each output line to be a sequence number. The format for sequence numbers must be adhered to if the output file is to be used with other programs on the Applied Logic system since the .WRITE statement affixes a CR-LF . It is always safe in this case to use a tab as the sixth character. The user who uses the .NWRITE statement should take care not to get a seven-character line consisting of five numerals, a tab, and a line-feed.

.NSNOUT

This statement turns off the feature initiated by the .SNOUT statement.

.PUSHJ L

This statement causes a transfer to L or in the case of inindirect addressing to the label named in L . The address of the statement immediately following a .PUSHJ statement is saved on a push-down list.

.POPJ

This statement pops the address pushed on by the last .PUSHJ statement and transfers to the address.

.POP

This statement is used to pop off the last address pushed on the push-down list by the last .PUSHJ statement. However, no transfer is made.

.MACRO

This statement is a signal to the SNOBOL compiler to consider subsequent lines of coding to be MACRO assembly code.

.SNOBOL

This statement is a countermand to the SNOBOL compiler revoking a previous .MACRO statement. Lines subsequent to this statement are considered by the SNOBOL compiler to be SNOBOL source. If the first character in a line is an up-arrow (↑), the line is considered to be MACRO code if .SNOBOL is currently in effect, or to be SNOBOL code if .MACRO is in effect.

.EXIT

This generates an exit call to the system, closes all open files, and results in the user being put in monitor command mode and the Teletype responding with

```
EXIT
↑C

.
```

.END

This statement signals the SNOBOL compiler that there is no more code to follow. If this statement is omitted, the compiler assumes an END statement. The .END statement generates a .EXIT statement.

SUMMARY OF SNOBOL FEATURES

LITERAL STRINGS

""	null string
"""	quote
"	aaa---a"

STRING ELEMENTS

"123"	literal strings
STR2	string names

SUBSTRING ELEMENTS

STR2	string names
"123"	literal strings
"123"!STR2	disjunctions of string elements
STR2	make STR2 be minimum length
STR2/n	make STR2 be length n
STR2/TEE	make STR2 (have length = integer value of TEE)
&STR2&	make STR2 be maximum length

STRING COMMANDS

```
LBL:  NAME   STRNG   = REP   /S(LBL1)F(LBL2); REMARKS full statement-
                              replacement.

      NAME   STRNG =  pattern search and delection.

      NAME←  STRNG    pattern search in Anchor mode

      NAME   SUB1]SUB2 = REP1                 }
      NAME   SUB1[SUB2]SUB3 = REP2            }  partial substring replacements
      NAME   SUB1[SUB2 SUB3 = REP2 REP3       }
```

SUMMARY - (cont'd.)

NORMALIZED SNOBOL INTEGERS

""	Zero
"ABC"	Zero
"7C"	7
"-7C"	-7
"ØØ34T7"	34
"A73"	Zero
"383"	383

INTEGERS OPERATIONS

```
+ - * /

NBR + "4"

NBR * SUM

"4" - "3"
```

TRANSFER COMMANDS

/ (LABEL)	unconditional
/S(LBL)	if search or relation is successful
/F(LBL)	if search or relation fails.
/S(LBL1)F(LBL2)	2-way branch
/F(LBL2)S(LBL1)	

ARITHMETIC RELATIONS

```
STR1   ==  STR2

STR1   #   "73"

A      <=  B

A      <   STR1

STR1   >   B

STR1   >=  STR2
```

SUMMARY - (cont'd.)

SPECIAL

; precedes a remark

↑ precedes a line of Macro code in SNOBOL mode

SPECIAL COMMANDS

.TYPE	$E_1\ E_2 \ldots E_n$	Type a string of characters plus <RETURN>
.NTYPE	$E_1\ E_2 \ldots E_n$	Type a string of characters.
.ACCEPT	S	Read from TTY into S until <RETURN>
.OPIN	S	Open the file named in S as the input file
.OPOUT	S	Create a file named in S and open it as the output file.
.APOUT	S	Open the end of the file named in S as the output file.
.WRITE	$E_1\ E_2 \ldots E_n$	Write a string of characters plus <RETURN> in the output file.
.READ		Read from the input file one record into S.
.CLIN		Close the input file.
.CLOUT		Close the output file.
.RIN	S	Close the input file and rename it as S.
.ROUT	S	Close the output file and rename it as S.
.SNIN		Set input mode to accept sequence numbers.
.NSNIN		Cancel the sequence no. input mode.
.SNOUT		Set output mode to write sequence numbers.
.NSNOUT		Cancel the sequence number output mode.
.PUSHJ	SUBR	Call subroutine named SUBR
POPJ		Return from subroutine.
.POP		Delete return address from last subroutine call.
.MACRO		Set mode to accept MACRO code.
SNOBOL		Return mode from MACRO to SNOBOL mode.
.EXIT		Exit from running program into ALC monitor command mode.
END		Denotes end of SNOBOL coding for a program.