

ELIZA IN SNOBOL4

Robert T. Duquet. Department of Computer Science, State University of New York at Albany, 1400 Washington Avenue, Albany, New York 12203, telephone 518/457-3300. 1970 November 25.

**ABSTRACT** When implemented in Snobol4, Eliza is dramatically shorter and simpler than the original Fortran program. The brevity of the rewritten Eliza greatly enhances its value in a course on artificial intelligence. The complete program, and a portion of a script are included as Appendices.

**Key Words and Phrases:** Artificial Intelligence, Snobol4, Eliza

**CR Categories:** 3.60, 1.52, 4.22

**INTRODUCTION** Eliza was an instant success when it was first introduced to computer society by Professor Weizenbaum (5). Although time may have diminished some of the original fascination of Eliza, it remains a delightfully interesting creature, particularly for students in an introductory course on artificial intelligence (1). To enhance this attractiveness, Eliza has been reprogrammed in Snobol4 (2). The result is much shorter and simpler than the original program and is therefore much more readily understood (providing, of course, that the students have some familiarity with Snobol4--an independently desirable condition!)

After a quick review of the basic Eliza algorithm, this article discusses the Snobol4 implementation, emphasizing those unique aspects of Snobol4 that made the language particularly suited for the task. Some major simplifications in the preparation of an Eliza script along with additional power and flexibility placed in the hands of the scriptwriter are discussed\*. The article concludes with a few miscellaneous comments concerning languages and techniques for artificial intelligence.

**THE ELIZA ALGORITHM** The objective of Eliza is to transform English sentences into other English sentences in such a way as to create an illusion of understanding and conversation. For this purpose the sentences may be regarded as mere strings of characters that are to be transformed to other strings according to the following steps:

1. The input is scanned for the occurrence of key words; i.e., for significant substrings.
2. The keys are inserted in a linear list according to a pre-assigned rank. Where appropriate, the key words are replaced in the input by alternate words.
3. The revised input string is examined for the presence of relatively simple patterns that, in normal English usage, commonly appear in conjunction with the highest-ranking key. If no such patterns are found, those associated with lower ranking keys are tried.
4. On a successful pattern match, a suitable portion of the context is concatenated with one of a number of stock phrases.
5. If no pattern is discovered in the current input sentence, a previous sentence may be re-used with one or more stock phrases that return the "conversation" to an earlier topic.

The key words, their substitutes, the associated patterns and the stock phrases are all read from a "script" before the "conversation" begins. The "personality" of Eliza may be completely altered by changing this script.

The original Eliza was written in Fortran, supplemented by the Slip package (6). It was heavily reliant on list-processing techniques; the script had to be cast as a list, the program converted each sentence to a list, and pattern matching was performed.

---

\*Extensions to Eliza described by Weizenbaum in (7) have not been included in the Snobol4 implementation.

ed by an analysis of list structures. This approach was entirely a matter of efficiency and convenience. (Convenience here means that Fortran was widely known and available, not that the program was particularly simple to write.)

#### A SNOBOL4 IMPLEMENTATION

##### The Basic Concepts

As indicated in the summary above, Eliza may be viewed as an exercise in string manipulation rather than list processing. Consequently, it is relatively straightforward to implement in a string-manipulation such as Snobol4. It turns out that fewer than 100 Snobol4 statements are required to program Eliza.

The Snobol4 language permits the creation of variables at execution time. Thus, if the program reads the character string XXXX into variable V, the statement \$V = 'ABC' automatically creates a variable named XXXX and assigns ABC as its value. Subsequent reference to \$V in an expression or in a pattern will refer to the string 'ABC'. In large measure, the simplicity of Eliza in Snobol4 is due to this indirect-addressing capability.

Each keyword in an Eliza script is used to create a set of uniquely-named variables that contain the pertinent information for that word. Thus, if XXXX is a keyword, the variable S.XXXX will contain the substitute (if any) for XXXX. Similarly, L.XXXX will contain the level number (rank) of the keyword, N.XXXX will contain the number of decomposition rules (patterns) associated with XXXX, D.i.XXXX (where i is an integer) will contain the i<sup>th</sup> such rule and R.i.XXXX will contain all the recomposition rules (context to be selected plus stock phrases) to be used when the i<sup>th</sup> decomposition is successful. Within the string represented by R.i.XXXX, individual recomposition rules are delimited by an arbitrarily-selected special character.

Keywords could be hash-coded and stored in an array as in the original Eliza implementation. Instead, for maximum simplicity, all keywords (suitably delimited) are concatenated in a single string. One pattern matching statement is all that is then required to determine whether a given word occurs as a substring of this "dictionary" string.

The keyword substitutes and the recomposition rules are stored as character strings. The level number and the tally of decomposition rules for a given keyword are stored as integers. The decomposition rules are stored as patterns (a special data type in Snobol4) that are constructed by the program from information supplied in the script.

**Decomposition Rules** In most cases, decomposition rules contained in a script will consist of a single string that is expected as a substring in input sentences. In these cases the program constructs a pattern in such a way that when the pattern matches an input sentence, a variable named POST will contain that portion of the sentence that follows the specified substring. The content of that variable may then be used in the recomposition procedure.

When the decomposition rule contains merely a null (an empty string) the pattern constructed by the program will unconditionally match any input sentence and the variable POST will contain that entire sentence.

Although the two patterns described above are rather special cases of the general decomposition procedures discussed in the original Eliza article, they serve about 90% of the cases shown in the original sample script. A more general mechanism is necessary, however, and is readily supplied by the Snobol4 primitive function, eval.

When a script contains a decomposition rule that starts with the word SNOBOL, the program recognizes that the remainder of that rule is a "pattern" in the sense (and syntax) of Snobol4. The eval function converts that part of the script (which has been read as a string) to a datum of the type "pattern".

When using this form of decomposition rule, the scriptwriter may assign any portion of the input sentence to any variables he may wish to create. For example,

a decomposition rule may be stated as follows:

```
SNOBOL arb . FIRST 'I' arb . SECOND 'YOU' rem . THIRD
```

where the first word, SNOBOL, serves to identify the remainder of the line as a decomposition rule of the second kind. This pattern matches any sentence that contains 'I' and 'YOU' (in that order) separated by any number of characters at any point in the sentence. The pattern assigns the portion of the sentence preceding 'I' to the variable FIRST, any portion of the sentence between 'I' and 'YOU' to SECOND and anything following 'YOU' to THIRD. FIRST, SECOND and THIRD are variables created by the script.

It is apparent that the use of eval has considerably expanded the type of decomposition rule available to the scriptwriter since all of the many Snobol<sup>4</sup> primitives and operators that facilitate the specification of patterns are now at his disposal. The amount of Snobol<sup>4</sup> language required to write a script is not large, however, since three operators (concatenation, alternation and conditional assignment) and a few primitives (arb and rem in the example above) will suffice for all but the most elaborate decomposition rules.

To avoid potential conflict between variables created and named by the scriptwriter and those program variables that should not be modified, all of the latter have a special character (a period) incorporated into their names.

Recomposition Rules As indicated previously, recomposition rules (prescriptions for a "response" to a given input sentence that has been successfully decomposed) are stored as strings. The strings contain stock phrases (literal substrings) and names of variables (such as POST or other scriptwriter-defined variables) whose contents are to be concatenated with the literals. The rules are followed by the program simply by applying the Eval function to the string wherein the rule is stored.

Actually, several recomposition rules are usually given for each decomposition. All rules in such a set are stored, suitably delimited, in one larger string. The rule selected in any instance is the leading one in the appropriate string. When it is used, it is shifted cyclically from the leading to the trailing position. The simplicity with which a string can be cut and re-assembled obviates the type of counter or pointer for each such set of rules that was used in the original Eliza implementation.

Whenever 'MY' becomes the dominant keyword in an input sentence, that sentence is stored for later use on those occasions when the input either contains no recognizable keys or cannot be decomposed successfully by any of the given rules. The set of sentences retained for this purpose is maintained as a string but used as a pushdown stack; i.e., previous sentences are recalled on a last-in-first-out basis. The word 'MY' was chosen for this special role solely because it was so used in the original Eliza. An alternative word or (or set of words) may be designated by the scriptwriter through redefinition of the variable, RETAIN. The variable CLUELESS contains stock phrases to be used as self-contained responses when all else fails.

SCRIPTWRITING General Scripts for the original Eliza were encumbered by many levels of parentheses used to cast the script as a suitably structured list. A script is now regarded merely as a string that is segmented into suitable substrings by a small set of metacharacters with reasonably high mnemonic value.

There are two types of substrings that we will designate as normal and exceptional elements. Normal elements begin with a keyword, contain information germane to that key and terminate any number of lines (cards) later with a metacharacter (a colon followed by a slash :/). Exceptional elements start with an asterisk, extend over only one line (card), and (usually) contain a Snobol<sup>4</sup> statement that is to be incorporated into the program. More on these exceptional elements later.

**Normal Elements** The normal element of a script contains one decomposition rule and the associated recomposition rules for the keyword that initiates the substring. To specify a number of decomposition rules for the same key, that word may initiate as many elements as the scriptwriter desires; these elements need not be contiguous in the script. The normal element may also contain a level number and a substitute for the keyword. To distinguish between types of information in an element, the information is enclosed in a pair of slashes (/) and is preceded by the single letter S (for Substitute), L (for Level number), or D (for Decomposition) delimited by one or more spaces. For example, MAYBE L /2/ D // CF PERHAPS:/ states that the keyword MAYBE has been assigned rank 2, the null decomposition rule matches any sentence, and the (single) recomposition rules refers the program to rules for the keyword PERHAPS.

**Exceptional Elements** Snobol<sup>4</sup> programs are interpreted rather than compiled or assembled. Capitalizing on this fact, Eliza allows the scriptwriter to include Snobol<sup>4</sup> statements that are to be executed as they are encountered. By this means, substring categories can be defined for later use in decomposition and/or recomposition rules. For example, the script line

```
*SNOBOL PARENTS = ('FATHER' | 'MOTHER') . PARENT
```

will create the variable PARENTS whose value will be a pattern that matches either the substring FATHER or the substring MOTHER. It also assigns whichever substring is actually matched to the variable PARENT. Subsequently, the decomposition rule

```
SNOBOL arb 'MY' arb PARENTS ' IS' rem . AFTER
```

and the recomposition rule

```
'WHY IS YOUR ' PARENT AFTER
```

applied to the input sentence 'I THINK MY OLD FATHER IS MAD' would produce the reply 'WHY IS YOUR FATHER MAD'.

This feature may also be used to debug a script since the script can include request to output specific variables, to initiate tracing or to perform any other desirable program modification. The end of a script is signalled by a line that begins with an asterisk and the words END OF SCRIPT.

**MISCELLANEOUS COMMENTS** By current standards Eliza is a rather trivial and outdated example of artificial intelligence (4), (7). That makes it all the more important to minimize the amount of effort required for a student to comprehend the operation of the program.

To date, the most commonly used language in artificial intelligence has been Lisp (3). Among the most important characteristics of Lisp are: The manner of representing and manipulating data structures; Dynamic and automatic allocation of computer memory; The ability to execute data as program statements. To a considerable degree, the facilities of Snobol<sup>4</sup> equal or surpass those of Lisp in the above respects. The student of artificial intelligence should be made aware of this alternate tool.

Earlier comments contrasted the original list processing basis for Eliza with a string manipulation approach. It should be understood, however, that the Snobol<sup>4</sup> interpreter employs list processing techniques to manipulate strings. The pre-eminence of these techniques in artificial intelligence is therefore not questioned.

A Fortran-based version of Eliza is undoubtedly more efficient than a Snobol<sup>4</sup> version. But the power of Snobol<sup>4</sup> and the economy of its notation have permitted the replication of Eliza in such a way that the fundamental algorithm stands clearly revealed in the program. For tutorial purposes at least, this clarity more than justifies the loss of computational efficiency.

## APPENDIX A - PROGRAM

```

* ELIZA IN SNOBOL      R T DUQUET      SUNYA      FALL 1969
*
  GANCHOR = 1          ; INPUT('INPUT',5,72)
  PRE.TRIM = SPAN(' ') ; NULL
  P.0 = *KEY.
  P.1 = PRE.TRIM BREAK(' ') . WORD. ' '
  P.2 = *LEN(NEWSIZE.) '/'
  P.3 = BREAK('::') . CONTENT. '::'
  P.4 = PRE.TRIM REM . CONTENT.
  P.5 = PRE.TRIM ANY('SLD') . WORD. ' '
  SHORTEN. = BREAK('.,') . PHRASE. ANY('.,') REM . TRAILER.
  X.REF      = PRE.TRIM 'CF'
  BUMP.      = PRE.TRIM 'NEWKEY'
  PAREN.     = PRE.TRIM '/' BREAK('/') . CONTENT. '/'
  STASH.     = PRE.TRIM '/' BREAK('/') . *$STORE. '/'
  CALL.TO.SNOBOL = PRE.TRIM 'SNOBOL'
  ATTENTION. = PRE.TRIM '*'

  INTRODUCTION = 'HOW DO YOU DO.'
  CLUELESS    = '... VERY INTERESTING'
  RETAIN      = 'MY'

  WE NOW READ THE SCRIPT AND FORM STRINGS AS FOLLOWS.
  FOR EACH KEY WORD 'XXXX' WE FORM THE FOLLOWING VARIABLES:
  * RPL.XXXX      IS A REPLACEMENT WORD.      (OPTIONAL)
  * LEV.XXXX      IS A LEVEL NUMBER (IF ABSENT KEY IS IGNORED)
  * N.XXXX        A COUNT OF THE NUMBER OF DECOMPOSITIONS
  * DEC.I.XXXX    IS THE I'TH DECOMPOSITION PATTERN
  * RULE.I.XXXX   IS A STRING OF RECOMPOSITION RULES FOR THE I'TH
                  DECOMPOSITION. RULES ARE SEPARATED BY '::'.

  OUTPUT      = 'ENTER SCRIPT'
  KEYWORDS.   = ' '
HIGGINS      SCRIPT. = TRIM(INPUT)              :F(END)
              SCRIPT. ATTENTION. =             :S(FLAG)
              SCRIPT. P.1 =                     :F(HIGGINS)
              KEY. = ' ' WORD.
              KEYWORDS. P.0                       :S(LESSON)
              KEYWORDS. = KEY. KEYWORDS.
LESSON        SCRIPT. P.5 =                     :F(HIGGINS)S($WORD.)
ERR           OUTPUT = 'SCRIPT ERROR: ' WORD. ' ' SCRIPT. :F(HIGGINS)
S            STORE. = 'RPL' KEY.
              SCRIPT. STASH. =                   :F(ERR)S(LESSON)
L            SCRIPT. PAREN. =                     :F(ERR)
              S('LEV' KEY.) = INTEGER(CONTENT.) CONTENT. :S(LESSON)
D            N.N = S('N' KEY.) + 1
              S('N' KEY.) = N.N
              SCRIPT. PAREN. =                     :F(ERR)
              CONTENT. CALL.TO.SNOBOL =           :S(SPECIAL)
              S('DEC.' N.N KEY.) = ARB CONTENT. REM . POST
RULES         STORE. = 'RULE.' N.N KEY.
              $STORE. = DIFFER(SCRIPT.) SCRIPT.    :F(NEW.CARD)
LOOP          NEWSIZE. = SIZE($STORE.) - 1
              $STORE. P.2                       :S(HIGGINS)
NEW.CARD      $STORE. = $STORE. TRIM(INPUT)        :F(END)S(LOOP)

```

```

*
*   THE FOLLOWING ARE SPECIAL SCRIPT-HANDLING STATEMENTS
*
SPECIAL  $('DEC.' N.N KEY.) = EVAL(CONTENT.)      ;(RULES)
ENCODE   SCRIPT. = SCRIPT. ' ;(HIGGINS) ;'        ;[CODE(SCRIPT.)]
FLAG     SCRIPT. CALL.TO.SNOBOL =                 ;S(ENCODE)
*
*   WE NOW HOLD A CONVERSATION.  FIRST WE READ A SENTENCE AND
*   SEARCH FOR KEY WORDS REPLACING APPROPRIATE ONES
*   AND STACKING THE KEYS IN A QUASI-ORDERED LIST (STRING).
*
INTRO     OUTPUT = INTRODUCTION
HEAR      PHRASE. = TRIM(INPUT) ' ;'                ;F(END)
HEARLESS  PHRASE. SHORTEN. ; PHRASE. = PHRASE. ' '
          COPY. = ; CUES. = ; CUE.LEVEL = 0
SPLIT     GANCHOR = 1 ; PHRASE. P.1 =              ;F(REPLY)
          GANCHOR = 0
          KEYWORDS. WORD.                          ;F(KEEP)
          NEW.WORD = DIFFER($('RPL.' WORD.)
                      $('RPL.' WORD.)              ;S(REPLACE)
          COPY. = COPY. WORD. ' '                  ;(STACK)
REPLACE   COPY. = COPY. NEW.WORD ' '
STACK     NEW.LEVEL = DIFFER($('LEV.' WORD.)
          $('LEV.' WORD.)                          ;F(SPLIT)
          CUE.LEVEL = GT(NEW.LEVEL,CUE.LEVEL)
          NEW.LEVEL                                ;F(LOCUE)
          CUES. = WORD. ' ;' CUES.                  ;(SPLIT)
LOCUE     CUES. = CUES. WORD. ' ;'                  ;(SPLIT)
KEEP      COPY. = COPY. WORD. ' '                  ;(SPLIT)
*
*   THIS PART FORMS OUR REPLY TO THE INPUT SENTENCE
*
REPLY     CUES. P.3 =                                ;F(NOCUE)
NEXTCUE   CUE. = ' ;' CONTENT.
          N.N = 0 ; NMAX. = $('N' CUE.)
ANALYSE   N.N = LT(N.N,NMAX.) N.N + 1                ;F(NOCUE)
          COPY. $('DEC.' N.N CUE.)                  ;F(ANALYSE)
          $('RULE.' N.N CUE.) P.3 =
          CONTENT. ' /' =
          $('RULE.' N.N CUE.) = $('RULE.' N.N CUE.) CONTENT. ' ;'
          CONTENT. X.REF =                          ;S(NEWCUE)
          CONTENT. BUMP.                            ;S(REPLY)
          OUTPUT = ' .. ' EVAL(CONTENT.)
          MEMORY. = IDENT(CUE., ' ;' RETAIN)
          LT(SIZE(MEMORY.),200)
          MEMORY. COPY. ' ;'                          ;(HEAR)
*
*   THIS IS WHAT WE DO IF THERE ARE NO KEY WORDS IN THE INPUT
*
NEWCUE    CONTENT. P.4                                ;(NEXTCUE)
NOCUE     PHRASE. = DIFFER(TRAILER.) TRAILER.        ;S(HEARLESS)
          MEMORY. P.3 =                              ;F(ER.AH.UM)
          OUTPUT = ' .. EARLIER YOU SAID ' CONTENT. ;(HEAR)
ER.AH.UM  OUTPUT = CLUELESS                          ;(HEAR)
*
END

```

## APPENDIX B - SCRIPT

```

* SNOBOL FAMI = 'MOTHER' ! 'FATHER' ! 'SISTER' ! 'BROTHER' ! 'DAUGHTER'
* SNOBOL FAM2 = 'MOM' ! 'DAD' ! 'WIFE' ! 'CHILDREN' ! 'HUSBAND' ! 'SON'
* SNOBOL FAMILY = (FAMI ! FAM2) . RELATIVE
* SNOBOL BELIEF = ('FEEL' ! 'THINK' ! 'BELIEVE' ! 'WISH') . PENSE
* SNOBOL HIGH = ('HAPPY' ! 'ELATED' ! 'GLAD' ! 'BETTER' ! 'HIGH') . BIEN
* SNOBOL ICKY = ('SAD' ! 'UNHAPPY' ! 'DEPRESSED' ! 'SICK') . MALADE
* SNOBOL MULTI = ('EVERYONE' ! 'EVERYBODY' ! 'NOBODY' ! 'NOONE') . ALLES
ALWAYS.      L /2/      D //
'CAN YOU THINK OF A SPECIFIC EXAMPLE ?':'WHEN ?':
'WHAT INCIDENT ARE YOU REALLY THINKING OF?':'REALLY, ALWAYS?':/
AM           L /2/ S /ARE/ D /ARE YOU/
'DO YOU BELIEVE YOU ARE' POST '?:
'WOULD YOU WANT TO BE' POST '?:
'YOU WISH I WOULD SAY YOU ARE' POST '?:CF WHAT:/
AM           D //
'WHY DO YOU SAY -AM- ?':I DONT UNDERSTAND THAT:/
ALIKE L /11/      D //CF DIT:/
ARE         L /2/ D /ARE I/
'WHY ARE YOU INTERESTED IN WHETHER I AM' POST 'OR NOT ?':
'WOULD YOU PREFER IF I WERENT' POST '?:
'PERHAPS I AM' POST 'IN YOUR DREAMS':
'DO YOU SOMETIMES THINK I AM' POST '?:?CF WHAT:/
ARE         D /ARE/
'DID YOU THINK THEY MIGHT NOT BE' POST '?:
'WOULD YOU LIKE IT, IF THEY WERE NOT' POST '?:
'WHAT IF THEY WERE NOT' POST '?:'POSSIBLY THEY ARE' POST:/
AUTRE      L /1/      D //
'PLEASE TELL ME ABOUT YOURSELF RATHER THAN ABOUT OTHERS':
'I AM MORE INTERESTED IN YOU THAN IN OTHERS.':
'DOES TALKING ABOUT OTHERS HELP YOU SOLVE YOUR PROBLEM?':/
BECAUSE    L /2/      D //
'IS THAT THE REAL REASON?':'DONT ANY OTHER QUESTIONS COME TO MIND?':
'DOES THAT REASON SEEM TO EXPLAIN ANYTHING ELSE?':
'WHAT OTHER REASONS MIGHT THERE BE?':/
CAN        L /2/ D /CAN I/
'YOU BELIEVE I CAN' POST 'DONT YOU?':
'DO YOU WANT ME TO BE ABLE TO' POST '?:
CF WHAT:'PERHAPS YOU WOULD LIKE TO BE ABLE TO' POST 'YOURSELF':/
CAN        D /CAN YOU/
'DO YOU WANT TO BE ABLE TO' POST '?:
'PERHAPS YOU DONT WANT TO' POST:
'WHETHER OR NOT YOU CAN' POST 'DEPENDS ON YOU, NOT ME':/
CAN'T S /CANT/
CERTAINLY L /2/      D //CF YES:/
COMPUTER  L /10/ D //
'DO COMPUTERS WORRY YOU?':'WHY DO YOU MENTION COMPUTERS?':
'WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM':
'DONT YOU THINK COMPUTERS CAN HELP PEOPLE':
'WHAT IS IT ABOUT MACHINES THAT WORRIES YOU':
'WHAT DO YOU THINK ABOUT MACHINES?':/
COMPUTERS L /10/ D //CF COMPUTER:/
DIT       D //
'IN WHAT WAY?':'WHAT RESEMBLANCE DO YOU SEE?':
'WHAT DOES THAT SIMILARITY SUGGEST TO YOU?':
'WHAT CONNECTIONS DO YOU SEE?':
'IS THAT THE REAL CONNECTION?':

```

```

'WHAT DO YOU SUPPOSE THAT RESEMBLANCE MEANS?':
'WHAT IS THE CONNECTION DO YOU SUPPOSE?':
'IS THERE REALLY SOME CONNECTION?': 'HOW?':/
DONT 'S /DONT/
DREAM      L /4/   D //
      'WHAT DOES THAT DREAM SUGGEST TO YOU?': 'DO YOU DREAM OFTEN?':
'WHAT PERSONS APPEAR IN YOUR DREAMS?':
'DONT YOU THINK THAT DREAM IS RELATED TO YOUR PROBLEM?': NEWKEY:/
DREAMED L /5/ S /DREAMT/   D //CF DREAMT:/
DREAMS   L /4/ S /DREAM/   D //CF DREAM:/
DREAMT   L /5/ D /YOU DREAMT/
      'REALLY, ' POST '?':
'HAVE YOU EVER FANTASIED ' POST ' WHILE YOU WERE AWAKE?': CF DREAM:/
EVERYBODY L /3/   D //CF EVERYONE:/
EVERYONE  L /3/ D /SNOBOL ARB MULTI REM . POST/
      'REALLY, ' ALLES '?': 'SURELY NOT ' ALLES '!': 'WHO FOR EXAMPLE?':
'CAN YOU THINK OF ANYONE IN PARTICULAR?':
'YOU ARE THINKING OF A VERY SPECIAL PERSON!':
'WHO MAY I ASK?': 'SOMEONE SPECIAL PERHAPS?':
'YOU HAVE A PARTICULAR PERSON IN MIND DONT YOU ?':
'WHO DO YOU THINK YOU ARE TALKING ABOUT?':/
HE        L /1/   D //CF AUTRE:/
HOW       L /2/   D //CF WHAT:/
I         L /2/ S /YOU/ D /YOU WAS/CF WAS:/
I         D /SNOBOL ARB 'YOU ' ('WANT ' ! 'NEED ') REM . POST/
      'WHAT WOULD IT MEAN TO YOU IF YOU GOT ' POST '?':
'WHY DO YOU WANT ' POST '?': 'SUPPOSE YOU GOT ' POST 'SOON?':
'WHAT IF YOU NEVER GOT ' POST '?': 'WHAT WOULD GETTING ' POST 'MEAN':
'WHAT DOES WANTING' POST 'HAVE TO DO WITH THIS DISCUSSION?':/
I         D /SNOBOL ARB 'YOU ARE ' ARB ICKY/
      'I AM SORRY TO HEAR YOU ARE ' MALADE:
'DO YOU THINK THAT COMING HERE WILL HELP YOU NOT TO BE ' MALADE '?':
'I AM NOT SURE IT IS NOT PLEASANT TO BE ' MALADE:
'CAN YOU EXPLAIN WHAT MADE YOU ' MALADE '?':/
I         D /SNOBOL ARB 'YOU ARE ' ARB HIGH/
      'HOW HAVE I HELPED YOU TO BE ' BIEN '?':
'HAS YOUR TREATMENT MADE YOU ' BIEN '?':
'WHAT MAKES YOU ' BIEN ' JUST NOW?':
'CAN YOU EXPLAIN WHY YOU ARE SUDDENLY ' BIEN '?':/
I         D /SNOBOL ARB 'YOU ' BELIEF ' YOU' REM . POST/
      'DO YOU REALLY THINK YOU' POST '?':
'BUT YOU ARE NOT SURE YOU' POST '?': 'DO YOU REALLY DOUBT YOU' POST '?':/
I         D /YOU ARE/
'IS IT BECAUSE YOU ARE' POST 'THAT YOU CAME TO ME ?':
'HOW LONG HAVE YOU BEEN' POST '?': 'DO YOU ENJOY BEING' POST '?':
'DO YOU BELIEVE IT NORMAL TO BE' POST '?':/
I         D /SNOBOL ARB 'YOU ' ('CANNOT' ! 'CANT') REM . POST/
      'HOW DO YOU KNOW YOU CANT' POST '?': 'HAVE YOU TRIED TO' POST '?':
'PERHAPS YOU COULD' POST 'NOW?':
'DO YOU REALLY WANT TO BE ABLE TO' POST '?':/
I         D /YOU DONT/
'DONT YOU REALLY' POST '?':
'WHY DONT YOU' POST '?': 'DO YOU WISH TO BE ABLE TO' POST '?':
'DOES THAT TROUBLE YOU ?':/
I         D /YOU FEEL/
'TELL ME MORE ABOUT SUCH FEELINGS.':
'DO YOU OFTEN FEEL' POST '?': 'DO YOU ENJOY FEELING' POST '?':

```



```

'WHAT DOES FEELING' POST 'REMIND YOU OF?':/
I      D /SNOBOL ARB 'YOU' ARB . POST 'I' /
      'PERHAPS IN YOUR FANTASY WE' POST 'EACH OTHER?':
'DO YOU WISH TO' POST 'ME?': 'YOU SEEM TO NEED TO' POST 'ME.':
'DO YOU' POST 'ANYONE ELSE?':/
I      D //
'YOU SAY' COPY '?:' 'CAN YOU ELABORATE ON THAT?':
'DO YOU SAY' COPY 'FOR SOME SPECIAL REASON?':/
I'M    L /2/ S /YOU ARE/ D //CF I:/
LIKE L /11/
LIKE D /SNOBOL ARB (' AM ' ; ' IS ' ; ' ARE ' ; ' WAS ') ARB 'LIKE'/
CF DIT:/
LIKE      D //NEWKEY:/
MACHINE'  L /10/ D //CF COMPUTER:/
MACHINES  L /10/ D //CF COMPUTER:/
MAYBE     L /2/  D //CF PERHAPS:/
ME S /YOU/
MY      L /3/ S /YOUR/ D /SNOBOL ARB 'YOUR' ARB FAMILY REM . POST/
      'TELL ME ABOUT YOUR FAMILY': 'WHO ELSE IN YOUR FAMILY' POST '?:'
'YOUR' RELATIVE '?:'
'WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR' RELATIVE '?:'/
MY      D /YOUR/ 'YOUR' POST '?:'
      RELATIVE '?:' D /YOUR/ 'YOUR' POST '?:'
'WHY DO YOU SAY YOUR' POST '?:' 'IS YOUR' POST 'IMPORTANT TO YOU?':
'DOES THAT SUGGEST ANYTHING ELSE THAT BELONGS TO YOU?':/
MYSELF S /YOURSELF/
NAME    L /15/ D //
      'I AM NOT INTERESTED IN NAMES': 'I'VE TOLD YOU BEFORE, I AM NOT
      INTERESTED IN NAMES - PLEASE CONTINUE':/
NO      L /2/  D //
'WHY -NO-?': 'WHY NOT?':
'ARE YOU SAYING NO JUST TO BE NEGATIVE?':
'YOU ARE BEING RATHER NEGATIVE!':/
NOBODY  L /3/  D //CF EVERYONE:/
NOONE   L /3/  D //CF EVERYONE:/
PERHAPS L /2/  D //
      'YOU DONT SEEM TO BE QUITE CERTAIN.': 'WHY THE UNCERTAIN TONE?':
'CANT YOU BE MORE DEFINITE?': 'YOU ARENT SURE?': 'DONT YOU KNOW?':/
REMEMBER L /6/ D /YOU REMEMBER/
      'DO YOU OFTEN THINK OF' POST '?:'
'DOES THINKING OF' POST 'BRING ANYTHING ELSE TO MIND?':
'WHAT ELSE DO YOU REMEMBER?': 'WHY DO YOU REMEMBER' POST 'JUST NOW?':
'WHAT IN THE PRESENT SITUATION REMINDS YOU OF' POST '?:'
'IS THERE A CONNECTION BETWEEN ME AND' POST '?:'/
REMEMBER D /DO I REMEMBER/
'DID YOU THINK I WOULD FORGET' POST '?:'
'WHY DO YOU THINK I SHOULD RECALL' POST 'JUST NOW?':
'WHAT ABOUT' POST '?:' CF WHAT: 'YOU MENTIONED' POST '!':/
REMEMBER D //NEWKEY:/
SAME     L /11/ D //CF DIT:/
SHE      L /1/  D //CF AUTRE:/
SORRY    L /2/  D //
'PLEASE DO NOT FEEL APOLOGETIC.': 'APOLOGIES ARE NOT NECESSARY.':
'WHAT FEELINGS DO YOU HAVE WHEN YOU APOLOGIZE?':
'I'VE TOLD YOU THAT APOLOGIES ARE NOT REQUIRED!':/
WAS      L /3/ D /WAS YOU/
      'WHAT IF YOU WERE' POST '?:' 'DO YOU THINK YOU WERE' POST '?:'

```

```

'WERE YOU' POST '?' : 'WHAT WOULD IT MEAN IF YOU WERE' POST '?' :
'WHAT DOES' POST ' SUGGEST TO YOU?' : 'CF WHAT' : /
WAS          D /YOU WAS/      'WERE YOU REALLY!' :
'WHY DO YOU TELL ME YOU WERE' POST ' NOW?' :
'PERHAPS I ALREADY KNEW YOU WERE' POST ' .' : /
WAS          D /WAS I/
'WOULD YOU LIKE TO BELIEVE I WAS' POST '?' :
'WHAT IF I HAD BEEN' POST '?' : 'WHAT SUGGESTS THAT I WAS' POST '?' :
'WHAT DO YOU THINK?' : 'PERHAPS I WAS' POST '!' : /
WERE          L /3/ S /WAS/   D //CF WAS : /
WHAT          L /2/   D //
      'WHY DO YOU ASK THAT?' : 'DOES THAT QUESTION INTEREST YOU?' :
'WHAT IS IT YOU REALLY WANT TO KNOW?' : 'WHAT DO YOU THINK?' :
'ARE SUCH QUESTIONS MUCH ON YOUR MIND?' :
'WHAT ANSWER WOULD PLEASE YOU MOST?' : 'HAVE YOU ASKED ANYONE ELSE?' :
'WHAT COMES TO MIND WHEN YOU ASK THAT?' :
'HAVE YOU ASKED SUCH QUESTIONS BEFORE?' : /
WHEN          L /2/   D //CF WHAT : /
WHY           L /2/   D /WHY DONT I/
'DO YOU BELIEVE THAT I DONT' POST '?' :
'PERHAPS I WILL' POST ' IN GOOD TIME!' : 'SHOULD YOU' POST ' YOURSELF?' :
'YOU WANT ME TO' POST ' DO YOU?' : 'CF WHAT' : /
WHY           D /WHY CANT YOU/
'DO YOU THINK YOU SHOULD BE ABLE TO' POST '?' :
'DO YOU WANT TO BE ABLE TO ' POST '?' :
'DO YOU BELIEVE OUR CONVERSATION WILL HELP YOU TO' POST '?' :
'HAVE YOU ANY IDEA WHY YOU CANT' POST '?' : 'CF WHAT' : /
WHY           D //CF WHAT : /
WON'T         S /WONT/
YES           L /2/   D //
      'YOU SEEM TO BE QUITE POSITIVE!' : 'ARE YOU SURE?' : 'I SEE!' :
'I UNDERSTAND.' : /
YOU           L /2/ S /I/   D /I ARE/
'WHAT MAKES YOU THINK I AM' POST '?' :
'DOES IT PLEASE YOU TO BELIEVE I AM' POST '?' :
'DO YOU SOMETIMES WISH THAT YOU WERE' POST '?' :
'PERHAPS YOU WOULD LIKE TO BE' POST 'YOURSELF?' : /
YOU           D /SNOBOL ARB 'I ' ARB . POST 'YOU'/
      'WHY DO YOU THINK I ' POST 'YOU?' :
'DO YOU LIKE TO THINK I ' POST 'YOU?' :
'WHAT MAKES YOU THINK I ' POST 'YOU?' :
'REALLY? I ' POST 'YOU?' : 'DO YOU WISH TO BELIEVE I ' POST 'YOU?' :
'SUPPOSE I DID ' POST 'YOU'. WHAT WOULD THAT MEAN?' :
'DOES ANYONE ELSE THINK I ' POST 'YOU?' : /
YOU           D /I/
'WE WERE DISCUSSING YOU - NOT ME !' : 'OH! I' POST '?' :
'ARE YOU REALLY TALKING ABOUT ME ?' : /
YOUR          L /2/ S /MY/   D /MY/
'WHY ARE YOU CONCERNED OVER MY' POST '?' :
'WHAT ABOUT YOUR OWN' POST '?' :
'ARE YOU WORRIED ABOUT SOMEONE ELSE'S' POST '?' : /
YOURSELF     S /MYSELF/
'SNOBOL INTRODUCTION = 'HOW DO YOU DO. PLEASE TELL ME YOUR PROBLEM.'
*   END OF SCRIPT

```

- REFERENCES
1. ACM Curriculum Committee. CURRICULUM 68. Comm ACM 11, 3. (March 1968) 151-197.
  2. Griswold, R.E. et al. THE SNOBOL<sup>4</sup> PROGRAMMING LANGUAGE. Prentice-Hall, Inc. (1968)
  3. McCarthy, J. et al. LISP 1.5 PROGRAMMER'S MANUAL. MIT Press (1965) 106 p.
  4. Simmons, R.F. NATURAL LANGUAGE QUESTION-ANSWERING SYSTEMS: 1969. Comm ACM 13, 1. (January 1970) 15-30.
  5. Weizenbaum, J. ELIZA--A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN AND MACHINE. Comm ACM 9, 1. (January 1966) 36-45.
  6. Weizenbaum, J. SYMMETRIC LIST PROCESSOR. Comm ACM 6, 7. (September 1963) 524-544.
  7. Weizenbaum, J. CONTEXTUAL UNDERSTANDING BY COMPUTERS. Comm ACM 10, 8. (August 1967) 474-480.

#### COMPILATION OF A SUBSET OF SNOBOL<sup>4</sup>

Paul J. Santos, Jr. and W. Douglas Maurer.  
University of California, Berkeley, Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, Berkeley, California 94720. This research sponsored in part by the National Science Foundation under grant GJ-821. 11 December 1970

This paper describes work we are doing on Snobol<sup>4</sup> compilation. The subset (Subbol) described here is an initial version intended to test some of the ideas involved, and a full version is now in the works, the compiler for which will be written in itself.

Subbol executes about 6.5 times faster than the Snobol<sup>4</sup> interpreter originating at the University of Maryland, both using the Univac 1108 65K Exec II operating system. It is anticipated that the full version will be even faster: Subbol has already been put to work at a local computer service bureau (Information Systems Design) to convert Fortran programs written for the GE635 into a form amenable to the Fortran/1108 compiler by doing such things as adding implicit do's in extra-long data statements and eliminating nonessential levels of equivalence variables.

Following is the abstract of a short paper covering our preliminary work which was presented at the Fourth Annual Princeton Conference on Information Sciences and Systems.

We here summarize some preliminary work done toward implementing a SNOBOL 4 system on the UNIVAC 1108 computer which is written in SNOBOL 4 and produces as its output an object program in UNIVAC 1108 symbolic assembler language, which can call or be called by FORTRAN or other assembler language programs. This represents a distinct advance in the technology of string processing compilers. In particular, it is hoped that users of SNOBOL who have written compilers for other languages in SNOBOL 4 will now be able to write efficient and economically feasible compilers for such languages in a compiled version of SNOBOL 4.

The SNOBOL string processing languages have proved very popular. SNOBOL 4 has almost always been implemented as an interpretive system, following the original.<sup>1</sup> Rumor has it<sup>2</sup> that an IBM 360 compiled SNOBOL 4 system exists; also according to rumor, this system is not written in itself, operates interpretively on patterns, and its object programs cannot be called by FORTRAN object programs.

The approach here is to remove permissive semantic features such as EVAL and CODE, and add a considerable number and variety of declarations. Default declarations are used whenever possible. Patterns are implemented as subroutines reflecting the structure of the pattern. Unevaluated expressions (the \* operator) can exist only inside patterns.

#### References

1. R. E. Griswold, J. F. Poage, and I. P. Polonsky, The SNOBOL 4 Programming Language, Prentice-Hall, New Jersey, 1968.
2. SIGPLAN Notices, Vol. 4, No. 11, November 1969, (describing a system written at IIT by Dewar and Belcher).