SNAP — An experiment in natural language programming

by MICHAEL P. BARNETT
H. W. Wilson Company
New York, New York

and

WILLIAM M. RUHSAM
Radio Corporation of America
Cherry Hill, New Jersey

INTRODUCTION

Computers are being used to a rapidly increasing extent, to manipulate and to generate materially, mechanically. (1) Many applications simply require items of information to be selected from a file of fixed format, heavily abbreviated records, and expanded into statements that are self-explanatory, and used perhaps in individual communications or incorporated in computer typeset compendia. At the other end of the spectrum are the interrelated challenges of mechanical indexing, abstracting, and translation. The burgeoning applications of computers to publishing, education, library work and information services in most major branches of science and scholarship are leading to a host of text processing and generating problems that span these limits of complexity.

Many of the programming problems of mechanical text processing also arise in other kinds of symbol manipulation, such as the compilation of computer programs, and the simplification of mathematical expressions. Several programming languages, such as COMIT and SNOBOL, have been developed over the years for mechanical symbol manipulation, and used to process text. FORTRAN, supplemented by some simple assembly language subroutines, has been applied to non-numeric problems quite extensively. ALGOL and COBOL also have been used this way. Several features of PL/1 will facilitate its use in processing text. A number of languages and processors have been developed for special kinds of text processing problems, such as mechanical editing.

SNAP is a procedural language for nonscientists to use. A SNAP procedure consists of a sequence of statements that have the appearance of simple English sentences. The primary rationale of SNAP is that mechanical text processing requires a language that many nonscientists will be willing to learn, and which they will find easy to use. Since such people deal primarily with English sentences in their daily work, a programming language that is a stylized subset of English should seem to them more "natural" than one that is symbolic. This rationale is not negated by the fact that nonscientists are using symbolic languages effectively. Many scientists once learned to write in assembly language, but the number of people programming scientific problems was greatly increased by FORTRAN.

Several SNAP constructions are quite like COBOL. SNAP is designed like BASIC, to enable a beginner to get useful results after learning very little, and to proceed by incremental learning efforts to deal with problems of increasing complexity. In its primary emphasis on string handling, however, SNAP is different; and in the details of the instruction set, and how the basic instructions are expressed, and in the proposed use of statements that invoke subroutines, to allow multitudinous language extensions, SNAP has some novel features.

A prototype processor for the basic SNAP language, that is SNAP without the subroutine capability, was implemented in a compatible subset of FORTRAN IV and now runs on several models of computers.
The language and the prototype processor are used in a graduate course on Computing and Librarianship at Columbia University. An early account of the SNAP project presented the language in a tabular form. (2) SNAP is used as a vehicle to develop basic programming concepts in a recent college text. (3) The implementation of the prototype processor is being reported too. (4) A more advanced processor, that will allow invoking statements and a range of definitional capabilities, is under development.

**Programming in MICROSNAP**

SNAP statements deal primarily with strings and quantities. A string can be given a name by a SNAP statement such as

```
CALL "YOU CANNOT ERR OR MAKE A BLOOMER WITH THE WARES OF ANCIENT SUMER" THE TAG.
```

This statement in a SNAP procedure makes THE TAG connote YOU CANNOT ERR, until another statement that redefines THE TAG is executed. In general, a CALL statement in SNAP (that is used to give names, not to invoke subroutines) consists of (1) the verb CALL, (2) an expression (for example a quotation) that specifies the string which is being given a name, (3) the name, and (4) a period. The forms that are allowed for the expression (2) are described in a later section.

SNAP places almost no restrictions on the choice of words that can be used as names. Extensive restrictions on the choice of names would seriously restrict the ease with which a "natural" programming language could be learned and used, and one objective of the SNAP syntax is to permit meta-words within names. How this may be done is discussed later. For the moment, we just prescribe that a name is a nonblank sequence of letters, digits, spaces, hyphens and apostrophes. Leading and trailing spaces are ignored, so are redundant internal spaces (i.e., any space that follows a space). An article (A, AN, THE) at the beginning of a name is optional (and in fact, ignored, to good advantage—see later).

A string can be printed by a SNAP statement that consists of (1) the verb PRINT, (2) a quotation, or a name that an earlier CALL statement gave to a string, or any of the other forms of string expressions that are to be described in a later section and (3) a period. The one word statement EXECUTE is used to end a procedure, and to make the processor start executing it, so that the SNAP conventions which have been described so far can be demonstrated by typing.

```
PRINT "'TESTING, TESTING'. EXECUTE.
```

or a little more adventurously, by typing

```
CALL "STILL TESTING" THE MESSAGE.
PRINT THE MESSAGE. EXECUTE.
```

The computer can be instructed to print output that is longer than the input, ready to be cut into two line display labels for the Mesopotamian Merchants Mart at the Roman Coliseum, by typing as follows:

```
CALL "YOU CANNOT ERR OR MAKE A BLOOMER WITH THE WARES OF ANCIENT SUMER" THE TAG. PRINT "CHARIOTS OF DISTINCTION". PRINT THE TAG. PRINT "INLAID GAMING BOARDS FOR PORCH AND PATIO." PRINT THE TAG. PRINT "DRINKING MUGS FOR THE LONGEST THIRST" PRINT THE TAG. EXECUTE.
```

The production of display labels that combine a common slogan, or class identification with individual identifications is a very simple and versatile "plot mechanism" for developing examples and exercises that relate to the interests of students in different disciplines. Others include

1. The production of sets of letters that consist of different selections of form paragraphs.
2. The production of programs for several performances of the same play or opera or other artistic work on different occasions, with perhaps some variation of case.
3. The production of a handbill that lists the events for an entire season in which a few different works are repeated many times.
4. The production of messages in large letters made of X's (GO DOG GO, keeps the burden of font design to a minimum), panoramic vistas of seagulls and palm trees on desert islands, processions of stylized animals, etc., printed along the length of the output stationery.

More are given in reference 3. The diversity of applications that can be handled with a minimal knowledge of SNAP has just been stressed because it does seem important to enable students of a nontechnical bent to overcome their initial apprehension of the computer by getting results before being overwhelmed by grammatical rules. Once this potential barrier is crossed, most seem able to assimilate programming grammar with ease. The kinds of examples cited here can, of
course, be handled with very little grammatical knowledge in many other languages.

Some more verbs that deal with strings

**Input statements:** The contents of a card, (or its image on magnetic tape, for off-line card input) is immittted by a SNAP statement such as

```
READ A BIOGRAPHICAL RECORD.
```

that consists of (i) the verb READ, (ii) the name that the user wishes to give to the string that is read, and (iii) a period. The corresponding instructions that begin with the verbs REQUEST and FETCH immit a input record from the console on which the user signed on (in installations where SNAP is used on line), and from all other input media respectively. In the latter case, the device (and the block format for magnetic tape input) is specified by a statement that begins with the word SELECT, and which remains in force until another SELECT statement that pertains to input is executed. The input commands are being extended to interface with operating system commands, to access files on backing storage conveniently. The REQUEST command alerts the console worker to type a record that is ended by pressing the line feed key; the SNAP processor stores the string under the name that is given in the command, and goes on to the next statement in sequence. There are further SNAP statements to instruct the processor to give certain characters a typographic control interpretation or to use them literally, and to retain trailing spaces in an input record, or to discard them. The latter provision is quite helpful in processes that embed an input item of variable length within a fixed text framework (e.g., a name in a form letter). An input record that consists of several items is deconcatenated by methods that are described later. The name that the input string is given by an input statement may have been used before, but does not need to have been.

**Output statements:** These consist of a verb, such as PRINT, followed by an expression that specifies the string to be recorded. This expression is, most simply, a quotation, or a name that was defined by an earlier statement in the procedure. Other forms are described later. The verb is PRINT, PUNCH, PERFORATE, TYPE and WRITE for the line printer, card punch, paper tape punch, console and all other media respectively. Instructions that begin with the word SELECT are used to control output on other media in a way that parallels their use for input. The SNAP conventions include provisions for representing case shifts, special characters and font changes in a more limited character set; an appropriate code conversion table can be put into the processor to record output on a device that has an extended typographic capability. A line printer with upper and lower case characters has been driven this way; so has a Flexowriter; and output has been recorded on a magnetic tape that then served as input to the composition programs of the RCA Videocomp electronic typesetting machine.

**String synthesis and alteration statements:** The CALL statement in general has the form

```
CALL b a.
```

where \( b \) denotes a string expression (i.e., an expression that displays or represents a string) and \( a \) denotes the name that this string is given by the statement. CALL statements are not recursive, that is the expression \( b \) must not make direct or indirect use of the name \( a \) (the APPEND statement mitigates this—see below). The definition that a CALL statement provides moreover is dynamic, that is, if the interpretation of the expression \( b \) changes after the CALL statement is executed, then the interpretation of the name \( a \) automatically changes to correspond. The COPY statement, of the form COPY \( b \) AND CALL IT \( a \) however constructs a copy, in core, of the string that \( b \) represents, and gives the name \( a \) to this copy. Subsequent changes in the interpretation of \( b \) do not affect the interpretation of \( a \).

The APPEND statement has the form

```
APPEND b TO a.
```

It gives the name \( a \) to the result of concatenating the string represented by \( b \) to the string known previously as \( a \).

The OVERWRITE statement has the forms

```
OVERWRITE b ON THE m-th AND (SUBSEQUENT, PRECEDENT) CHARACTERS OF a.
```

Here \( m \)-th denotes an ordinal adjective of the kinds such as 3-RD, and UMPTEEN-TH which are discussed later. A single character can be overwritten by a shorter form of the statement.

```
OVERWRITE b ON THE m-th CHARACTER OF a.
```

The DELETE statement, which elides characters, takes the forms

```
DELETE b FROM a.
```
DELETE THE \((m\text{-th}, m\text{-th} \text{ THROUGH } n\text{-th}, \ m\text{-th AND PRECEDENT, } m\text{-th AND} \text{ SUBSEQUENT CHARACTER(S) OF } a\).

The string name \(a\) in an APPEND, DELETE or OVERWRITE statement must have been given to a string by a COPY or an input statement previously (and more recently than by any CALL instruction). Expressions that purport to represent strings but which are inconsistent or invalid are considered to represent null strings.

Storage allocation: A SNAP procedure can be written without consideration of the lengths of the strings that are involved, subject to the total core storage capacity of the computer. The prototype processor allows 16K bytes in a 128 K byte RCA Spectra or IBM 360 computer, for strings and internal representations of procedures that are co-resident.

Procedures are condensed appreciably in their internal representation (in “SNAPIC” code), so that for many problems there is ample room for all the strings involved, without recourse to paging tactics; and these can be adopted, using backing storage, when need occurs.

Strings that are defined by CALL statements are represented internally by codes within the actual representations of the statements; but strings which are immittted by input statements, or constructed by COPY, APPEND, DELETE and OVERWRITE statements are stored separately in a“string bank.” When a COPY or an input statement is executed, that assigns a string to a particular name for the first time in the current operation of a procedure, this string is stored sequentially in the unused portion of the string bank. When the string that is known by a particular name is changed, the space that it occupies is used for the new string, and chained to a disjoint portion of the string bank if it is inadequate. Since sequentially stored material can be processed more rapidly than disjoint material in many circumstances, SNAP includes an instruction of the form

\[
\text{RESERVE SPACE FOR } n \text{ CHARACTERS IN } a.
\]

where \(n\) stands for a positive integer, and \(a\) for a string name, that may have been used before, but need not have been. This reserves a continuous portion of the string bank, that is \(n\) characters long, for strings called \(a\). It does not preclude longer strings receiving this name—they are simply chained. The statement may be advantageous when the strings that are given the name \(a\) vary in length during a procedure, and it is possible to anticipate a value which this length is unlikely to exceed, or to impose a limit on this length. Storage allocation thus is permitted to the user, but is not imposed on him.

Instructions that deal with integers

A statement such as

\[
\text{SET I TO 7.}
\]

that consists of (1) the verb SET, (2) a mnemonic, word or phrase that the user chooses for a particular quantity, (3) the preposition TO, (4) an integer, and (5) a period, has the dual effect of giving the status of a quantity name to the word(s) or mnemonic that appears between SET and TO, and assigning the value of the integer (item (4)) to this name, until further statements change it. More generally, the item (4) may be either

(i) an integer
(ii) a quantity name that was introduced by an earlier SET statement,
(iii) a length expression such as \(\text{THE LENGTH OF } b\), where \(b\) stands for an expression that represents a string (expressions for lengths of lists are discussed later)
(iv) an arithmetic expression of the form

\[
\text{THE } f \text{ OF } e_1 \text{ AND } e_2
\]

where \(f\) denotes one of the words SUM, DIFFERENCE, PRODUCT, QUOTIENT, REMAINDER, CEILING, GREATER, LESSER; \(e_1\) denotes an integer or a name that was introduced by an earlier SET statement, and so does \(e_2\). A name can be used more than once in a SET statement. An article (A, AN, THE) is optional at the beginning of a quantity name and it is ignored when it is included. Invalidity is infectious, that is a quantity defined in terms of an invalid quantity is invalid.

(v) a string expression that represents a string which is a decimal integer, with perhaps redundant leading zeroes, a sign, and leading and trailing spaces.

Defining lists of strings and quantities

A list of strings can be defined by a statement such as CALL “SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY” THE DAY LIST. This permits subscripted names, such as THE 1-ST DAY and THE 5-TH DAY to be used for strings, in any of the ways that unsubscripted string names (e.g., THE TAG used earlier)
can be used. A list element thus can be redefined individually, by CALL, COPY, APPEND, DELETE and OVERWRITE statements. It can be recorded by an output statement, and used in expressions that define further strings. A quantity name that has been defined in any of the ways that were described earlier can be used as a symbolic ordinal, by adding -TH, so that if N and UMPTEEN are quantity names, THE N-TH DAY and THE UMPTEEM-TH DAY are acceptable as subscripted string names.

In general, a CALL statement of the form

\[
\text{CALL} \left( s_1, s_2, \ldots, s_k \right) \text{ THE } g \text{ LIST.}
\]

gives the status of a generic string name to the word or phrase that is denoted by g, and permits expressions of the form THE j-th g to be used as subscripted string names where j-th denotes a numerical ordinal (e.g., 1-ST, 2-ND, 73-RD) or a symbolic ordinal (e.g., N-TH.) A subscripted string name is interpreted as a null string if the ordinal is invalid, or if its value is inappropriate. A comma is forced in a list element by a preceding asterisk, and individual elements are defined to be null by adjacent delimitors (commas and/or quote marks.)

A generic string name also can be introduced by input statements of the form

\[
\begin{align*}
\text{(READ, REQUEST, FETCH)} & \ (A, AN, THE) \\
& \ g \text{ LIST.}
\end{align*}
\]

This initiates a record, and treats commas as separators between list elements (except when forced by a preceding asterisk). A generic string name also can be introduced by a statement of the form

\[
\text{RESERVE SPACE FOR } k \text{ STRINGS IN THE } g \text{ LIST.}
\]

where k denotes a positive integer. The same name can be given to lists of different length, on different occasions in the execution of a procedure, and the processor accommodates these changes automatically by chaining. The RESERVE statement may be used to take advantage of knowledge of a limit that is likely, or which can be imposed.

An entire list can be recorded in the output by a statement of the form

\[
\begin{align*}
\text{(PRINT, PUNCH, WRITE, PERFORATE, Type) THE } & \ g \text{ LIST.}
\end{align*}
\]

Successive elements are separated by commas, which are adjacent for elements that are null. Trailing null elements and their commas however are suppressed. Trailing null elements also are ignored in statements of the form

\[
\text{SET } k \text{ TO THE LENGTH OF THE } g \text{ LIST.}
\]

A further statement, however, of the form

\[
\text{SET } k \text{ TO THE REGISTERED LENGTH OF THE } g \text{ LIST.}
\]

that includes trailing null elements in the list, as it was defined most recently, will be provided for programmed storage control.

A list of numbers n₁, n₂, ..., nₖ can be defined by a statement of the form

\[
\begin{align*}
\text{SET THE } & \ h \text{ LIST TO } n_1, n_2, \ldots, n_k.
\end{align*}
\]

Expressions that represent strings

A string expression, that is an expression which displays or represents a string, may take any of the following forms in a SNAP statement.

1. A quotation, that is bounded by quote marks. Within a quotation, /, $, 1, >, and < signify forced line break, forced page break, case reversal, upper case and lower case respectively. An asterisk is typed before one of these characters, or a quote mark, to force its literal use within a quotation. Two asterisks are typed to represent a single literal asterisk. An = symbol together with the character that follows represents a special character. When a quotation continues from one input card (line) to the next, one space is included between the last non-blank character on the first card, and the first character on the next, except when the former character is a hyphen, in which case it is elided and the space is not included.

2. An unsubscripted string name, that was introduced by a CALL, COPY, input or RESERVE
statement, at an earlier point in the procedure, both as written and as executed.

3. A subscripted string name that contains (i) a numerical ordinal, or a symbolic ordinal derived from an unsubscripted quantity name that was defined previously; and (ii) a generic string name that was introduced by a CALL, input or RESERVE statement which ends with the word LIST, at an earlier point in the procedure.

4. An integer that is positive, or negative, or zero. Leading zeroes, a plus sign, and spaces before and after the integer in the expression are elided. This is because the integer is first stored as a quantity, and then converted back to a string representation. The statement PRINT 007, thus makes the computer print 7 in the first type position. The statement PRINT "007," however makes the computer print 007 in type positions 1 to 3, since 007 is stored as a string because of the encompassing quote marks.

5. An unsubscripted quantity name that was introduced by an earlier SET statement. This is interpreted, in a context that requires a string expression, as the string of characters that represents the value of the quantity, without any leading spaces, or redundant zeroes, or a sign when it is positive.

6. A subscripted quantity name that contains a generic quantity name which was introduced by an earlier SET ... LIST ... or RESERVE statement. This is treated in the same way as (5).

7. An extract expression of the form

THE k-th CHARACTER OF a.

or

THE k-th THROUGH j-th CHARACTERS OF a.

where k-th stands for a numerical ordinal, or a symbolic ordinal that contains a previously defined unsubscripted quantity name; and j-th does too, and a denotes a string name.

8. A concatenated string expression, that consists of two or more items of the kinds described above, joined by the word THEN, to connote concatenation of the strings that they represent, within the entire string that the expression represents.

Control and conditional statements

A SNAP statement may be preceded by a bracketed label. This label may be cited in control statements of the form

(REPEAT, CONTINUE) (WITH, FROM) g.

where g denotes the label. The verbs REPEAT and CONTINUE are used respectively when g is earlier and later in the written procedure, for external appearances; but they are synonymous as far as the SNAP processor is concerned. The statements

REPEAT FROM THE (BEGINNING, BEGINNING OF THE PROCEDURE).

send control back to the first statement of a procedure, which does not need to be labelled. The one word statement

TERMINATE.

returns control to the operating system.

Two forms of conditional statement are used:

IF u v, OTHERWISE w.
IF u v.

The letters u, v and w here denote the condition, the success action, and the fail action respectively. The success action consists of one or more clauses, that could stand by themselves as unconditional SNAP sentences. The clauses are separated by commas, when there are several, and the word AND allowed between a comma and the first word of a clause. The success clause, when there is only one, and the last success clause, when there are several, may be of the (REPEAT, CONTINUE) (FROM, WITH) kind described above. Alternatively, it may be

CONTINUE WITH THE NEXT SENTENCE.

This is implied when the success action does not specify a transfer of control.

The fail action may be constructed in just the same ways as the success action, except that

CONTINUE AS FOLLOWS

is used as the final (or only) clause to take the next sentence in sequence. It is implied when the fail action does not specify a transfer of control, and as the entire fail action in the short form IF u v.

SNAP allows the following forms of condition clause at present:
1. THE INPUT IS EXHAUSTED
2. \( \xi \) IS (GREATER THAN, GREATER THAN OR EQUAL TO, EQUAL TO, LESS THAN OR EQUAL TO, LESS THAN, UNEQUAL TO) \( \zeta \)
3. \( s_1 \) (IS, ARE) \( \{ \) THE SAME AS \( \} s_2 \)
4. \( s_1 \) IS THE SAME AS THE \( m \)-th AND (PRECEDENT, SUBSEQUENT) CHARACTERS OF \( s_2 \)
5. THE \( m \)-th AND (PRECEDENT, SUBSEQUENT) CHARACTERS OF \( s_1 \) ARE THE SAME AS \( s_2 \).

\( \xi \) and \( \zeta \) denote quantity expressions, \( s_1 \) and \( s_2 \) denote simple string expressions of the kinds (1) to (7) listed previously. \( m \) denotes a numerical or a symbolic ordinal that contains a previously defined unsubscripted quantity name.

Some more examples

The account of SNAP in the last few sections covers almost all the features of the basic language. Several of these may be illustrated by the production of a simple calendar of the form

\[ \begin{array}{c}
\text{WEDNESDAY} \\
1 \\
\text{JANUARY} \\
1969 \\
\text{THURSDAY} \\
2 \\
\text{JANUARY} \\
1969 \\
\end{array} \]

A SNAP procedure to print this is as follows.

CALL "JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER" THE MONTH LIST.


CALL "SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY" THE DAY LIST.

SET M TO 1. SET N TO 4. SET K TO 1. SET J TO 1.

(PRINT A PAGE ACTION) PRINT THE N-TH DAY. PRINT "1969/".

PRINT THE M-TH MONTH. PRINT "1969/".

IF K IS EQUAL TO 365 TERMINATE, OTHERWISE CONTINUE AS FOLLOWS.

INCREASE K BY 1. IF J IS EQUAL TO THE M-TH LIMIT INCREASE M BY 1, AND SET J TO 1, OTHERWISE INCREASE N BY 1. REPEAT FROM THE PRINT A PAGE ACTION.

EXECUTE.

This example is quite useful as an illustration of the computer's ability to print much more than the user types, by repeating items in different combination. It can be extended and modified in numerous ways, with minimal knowledge of SNAP, which is useful for teaching proposes.

Another useful example, whose logic is a trifle more elaborate, reads a classification scheme from a deck of cards, such as

\[ \begin{array}{c}
\text{VERTEBRATA, (MAMMALIA, (PRIMATES, (ANTHROPOIDEA, (SIMIIDEA, CERCOPITHECIDAE, CEBIDAE, H APALIDAE), LEMUROIDEA, (LEMURIDAE, LORISIDAE, TARSIIIDAE, CHROMIDAE)), CHIROPTERA, (MICROCHIROPTERA, (VESPERTILIONIDAE, RHINOLOPIDAE, PHYLLOSTOMATIDAE), MEGACHIROPTERA, (P TEROPODIDAE)), INSECTIVORA, ((ERINCEIDAE, TALPIDAE, SORICIDAE, MACROCHIRIDAE, DIDODONTIDAE, URANOSCOPIDAE)))}} \\
\end{array} \]

that is strung out to save card space, with brackets indicating subordination, and prints it in a hierarchically indented format, that is

\[ \begin{array}{c}
\text{VERTEBRATA} \\
\text{MAMMALIA} \\
\text{PRIMATES} \\
\text{ANTHROPOIDEA} \\
\text{SIMIIDEA} \\
\text{CERCOPITHECIDAE} \\
\text{LEMUROIDEA} \\
\text{CHIROPTERA} \\
\end{array} \]

for the zoological scheme just cited. The procedure is
as follows

CALL "" THE BACKGROUND
SET P TO 1. SET Q TO 1.
CALL THE NULL STRING THE CARRY-OVER.

(INPUT TEST)
SET J TO 0. SET I TO 0. SET K TO 0.
IF THE INPUT IS EXHAUSTED CONTINUE WITH THE LAST ITEM ACTION, OTHERWISE CONTINUE AS FOLLOWS.
READ AN INPUT RECORD,
(COMPONENT TEST)
INCREASE J BY 1. CALL THE J-TH CHARACTER OF THE INPUT RECORD THE KEY.
IF THE KEY IS "("CONTINUE WITH THE DESCENT, OTHERWISE CONTINUE AS FOLLOWS.
IF THE KEY IS "," CONTINUE WITH THE OUTPUT ACTION, OTHERWISE CONTINUE AS FOLLOWS.
IF THE KEY IS ")" CONTINUE WITH THE ASCENT, OTHERWISE CONTINUE AS FOLLOWS.
IF THE KEY IS " " CONTINUE WITH THE LAST ITEM ACTION, OTHERWISE CONTINUE AS FOLLOWS.
CONTINUE WITH THE END CARD TEST.
(DESCENT) INCREASE P BY 1. INCREASE I BY 1. SET Q TO P.
CONTINUE WITH THE END CARD TEST.
SET X TO J. INCREASE X BY 1. SET I TO X. SET K TO J. SET Q TO P.
CALL THE NULL STRING THE CARRY-OVER. CONTINUE WITH THE END CARD TEST.
(ASCENT) DECREASE P BY 1.

(END CARD TEST)
IF J IS LESS THAN 80 REPEAT FROM THE CHARACTER TEST, OTHERWISE CONTINUE AS FOLLOWS.
COPY THE I-TH THROUGH K-TH CHARACTER OF THE INPUT RECORD AND CALL IT THE CARRYOVER. REPEAT FROM THE INPUT TEST.
(LAST ITEM ACTION)
PRINT THE 1-ST THROUGH Q-TH CHAR-
ACTER OF THE BACKGROUND THEN
THE CARRYOVER THEN THE I-TH
THROUGH K-TH CHARACTER OF THE
INPUT RECORD.
TERMINATE. EXECUTE.

The procedure can be shortened slightly by omitting OTHERWISE CONTINUE AS FOLLOWS from several IF statements. Hundreds of other examples of SNAP procedures are given in reference 3.

The prototype SNAP processor

The SNAP language was defined almost completely in late 1966. The processor was implemented in stages, in part because there seemed good reasons to demonstrate a working subset, and incremental progress, as quickly as possible, and in part because of uncertainty in the potential of the language, and in some of the details that might be needed. A processor that dealt with CALL and PRINT statements was developed first; input, COPY and unconditional transfer of control statements were added next; then arithmetic operations and conditional statements; and then the further string instructions, and the statements that deal with lists. The prototype processor is written almost entirely in FORTRAN IV. Implementation was started using a time shared PDP 6 computer. After a few months the processor was transferred to an RCA Spectra 70-45, and to the IBM 7094 at the Columbia University Computing Center, where class exercises were run for a semester using the interim version, while implementation was extended on the Spectra. Work on the prototype was ended recently. It is operating at present on several Spectra computers, and on the IBM 360-75/50 system at Columbia University; and a somewhat earlier version compiled and run on a UNIVAC 1108.

The prototype processor consists of (1) a small control section, (2) the translator, and (3) the interpreter. The translator immitis a SNAP procedure, and forms a numerical representation (in "SNAPIC" code) in the "procedure table". The interpreter then executes the processes that the SNAPIC representation specify. The control section simply calls the translator and the interpreter, and returns control to the operating system when appropriate.

The SNAPIC representation maps a verbal SNAP procedure fairly closely. Integers in different numerical ranges are used for command words, delimiters and precedence codes for different kinds of expression, and pointers to tables that contain, or point to, objects of interest. An un subscripted string name in the direct
object of a SNAP statement in SNAPIC is represented by a pointer to the “string directory”. The corresponding entry in this directory points to a definition of the name in the procedure table, or to the origin of the string in the string bank, depending on whether a CALL or a COPY or input statement that ends with the name was executed more recently. A quantity name is represented by a pointer to the “quantity bank” in which actual numerical values are stored. A subscripted string name is represented by the ordinal (subscript), and a pointer to the string list directory which points in turn to the entry for the first element of the list, in the subscripted string directory. This contains pointers that identify the actual strings which are elements of a list, in the same way that string directory elements identify the strings that are known by unsubscripted names. Pointers to successive elements of a list are stored consecutively in the subscripted string directory whenever possible; chaining is used when necessary. A numerical ordinal is represented by its numerical part, and a symbolic ordinal is represented by the negative of the appropriate pointer to the quantity bank. A subscripted quantity name is represented by the ordinal, and a pointer to the quantity list directory, which points in turn to the origin of the list in the subscripted quantity bank, that contains the actual values of the elements.

The translator was written in an ad hoc fashion. The initial, and incremental capabilities were needed, and obtained, in less time than could be spared to implement a reasonably powerful syntactic analyzer. An analyzer will be used in the translator of the advanced system that is being designed, and additional instructions will be provided to permit users to apply it to data strings at object time.

The control section, translator and processor occupy 56K, 45K and 48K bytes respectively, of which the last two can be overlayed, in the link edited version that runs on the Spectra 70-45. As an indication of the length of SNAPIC representation, the two procedures in the preceding section require just under 600 and 700 bytes respectively.

**SNAP as a teaching vehicle**

Programming in SNAP can be taught to non-scientists by introducing the basic constructions, and showing some of their uses, in the following sequence.

1. **MICROSNA P**: This subset of SNAP consists of (i) PRINT and (ii) CALL statements in which strings are displayed as quotations, or referenced by unsubscripted names, and (iii) EXECUTE statements, to start execution. Some exercises for which MICROSNAP suffices were described earlier in this paper.

2. **MINISNAP**: This subset of SNAP consists of the elements of MICROSNAP and the word THEN. It extends the variety of display labels, form letters, programs for the performing arts and for athletic and sporting events, catalog and greeting cards, and other materials whose mechanized production can be introduced by MICROSNAP, to allow different items to be joined on a line. This has an obvious benefit, for example in the production of form letters.

The addition of the word THEN, moreover, allows the introduction of several relatively general ideas. The production of most hierarchically structured materials (such as a set of catalog cards in which a journal name is repeated throughout while the details of the issue are repeated with infrequent change, and the details of the individual papers are repeated for just a few cards each; or the verbalizations of a long sequence of year numbers) can use hierarchical naming in a variety of ways, particularly when full use is made of the "implied redefinition" characteristic of the CALL statement. Constructing the shortest procedure that is possible for an application, to reduce keyboard work to a minimum, introduces the idea of optimization, in a way that the students can readily appreciate, and which has practical importance when a large volume of material is processed.

The problem of deciding which pieces of the output should be given names during its synthesis, and defining these names most concisely, using just CALL, THEN, quotations and other names, provides a challenge to the ingenuity of the student, after negligible grammatical instruction, that many non-scientists find novel and intriguing. Such examples, moreover, give the student an opportunity to develop an intellectual process, which he can then analyze, and reduce to an algorithm, with the prospect of mechanizing this after learning more programming grammar. An incentive is provided to consider formal descriptions of the structure of strings, which are amenable to simple algebraic manipulations, of potential relevance to the design of large files of data, and to some topics in stylistics, and to learn a little about the elementary uses of graphs.

Some simple combinatorial examples, such as the production of fifteen menus for table d'hote luncheons, that combine one of four appetizers, fish, meat and dessert dishes in all possible ways, can be handled by procedures that are shorter than the output they produce, and which can be generated by procedures that are even shorter still. Such examples help make the student conscious almost from the outset of the course, that procedures can be written to generate procedures, to advantage.
Some mechanized aspects of teaching also can be broached, using MINISNAP. Thus, given an output device with a reasonable typographic capability, the tactics that can be used to print personalized form letters can be applied to the production of a set of texts, that present MINISNAP (or anything else) to reader groups of different professional interests (e.g., middle XVth century armorial bearings, later XVth century armorial bearings, 4.2 mev nuclear physics, 4.25 mev nuclear physics), by substituting examples and exercises of specialized interest in a common explanatory framework. Another educational topic that may merit exploration is the mechanical production of a large number of exercises (from a relatively short prescription) that require the use of different combinations of elementary programming constructions, or mathematical manipulations, or equivalent operational units in other subjects. It is possible that some students might develop their "intuitions" for an activity which involves selecting and manipulating words or symbols, by working large numbers of exercises that could be generated this way, with a saving of human effort, and perhaps analyzed and criticized mechanically. The problem of generating exercises for SNAP starts to approach an interesting level of complexity by the time all the possibilities of MINISNAP are considered. It becomes interesting when the READ instruction is added, which comes next in the progression that is being discussed.

(3) MIDISNAP: The addition of the READ statement and REPEAT FROM THE BEGINNING to the elements of MINISNAP permits the separation of procedures and data, and some very elementary data driven procedure generating procedures. Adding extract expressions permits the deconcatenation of input records, and the expansion of fixed field records that omit characters (e.g., decimal points, units of measurement, century digits) which are implicit in the record design, to include these in the output. It is convenient to introduce IF statements that compare strings and statement labels next, then the COPY statement, and then arithmetic. By this stage, additional constructions, and applications are accepted as more or less expected matters of detail. MIDISNAP, that contains the elements which have just been mentioned, is adequate for a considerable range of text generating problems, that use fixed field files on punched cards (or with the addition of SELECT and FETCH, magnetic media); and may well be a convenient subset of SNAP for an elementary course in mechanized text processing and programming concepts, for implementation on relatively small computers.

(4) Basic SNAP: The further SNAP constructions that deal with strings and lists open the floodgates of programming rhetoric and applications. Containing these requires a subprocedure capability, such as the one considered next.

Scanning and invocation

SNAP is not the first programming language to be put into use without a subroutine capability, but with the hope that one would be added later. Plans for an extended SNAP processor are well advanced at the time of writing, that will permit subprocedures to be written in SNAP, and invoked by statements that increase the "naturalness" of the language considerably. Quite extensive experimentation probably is needed, with a working system, in working environments, to determine the relative benefits and hazards of the different paths along which the syntax of invoking statements can be developed. The extended processor is being designed to facilitate such experiments, by using a syntax driven translator to produce a canonical representation in extended SNAPIC, for the interpreter to execute.

One line of exploration, that seems to be particularly interesting, would continue to restrict the contexts in which new names for strings, and quantities, and lists, are introduced in procedures and subprocedures. Basic SNAP requires these to be introduced by CALL, COPY, input, RESERVE and SET statements at points which precede their earliest use, in the order in which the procedure is written, to define or alter other strings and quantities. It is possible therefore to scan a CALL statement from left to right alternately for (i) simple string expressions of the forms (1) to (7) given earlier, and (ii) the word THEN, and to treat the balance of the sentence that remains when THEN is not found as the name that the statement confers on a string (or on a list of strings). This name is added to the name table if it has been used before. The tactic can be elaborated, for example to allow names that are substrings of other names. Examples can be constructed to confuse the tactic, probably no matter how much it is elaborated, but the objective of natural language programming is to deal with statements that look "natural" rather than bizarre. The preposition TO can be allowed in the names of quantities (and strings) by using similar tactics in a right to left scan of SET statements.

The tactic can be extended fairly simply to procedures that contain invoking statements of the following kinds:

1. The statement begins with a word that is neither one of the basic verbs of SNAP nor IF; its only arguments are previously defined names, quotations and numbers; the framework in which these are embedded identifies the subprocedure; no
continuous piece of this framework is used as a name in the procedure.

2. The statement begins with a basic verb of SNAP, and contains one or more function expressions, in contexts in which the representation of a string or quantity is appropriate. Each function expression consists of an opening word or phrase that is characteristic of the function, followed by one or more arguments, of the kinds mentioned in the preceding paragraph, in alternation with further words, phrases and/or punctuation that completes the framework which identifies the function. It is convenient to allow either an argument or part of the framework to come last, and to require a comma as the last character in the latter instance. No continuous part of the framework may be a name that is used in the procedure, or a number, and nesting is prohibited.

3. The statement consists of a clause of the form (1) above, a comma, and a further clause such as FORMING X, Y, AND Z CONCURRENTLY, that in general consists of a participle ending in ING, one or more names that need not have been used previously, separated by commas and possibly AND; and a final word (e.g., THEREBY, ACCORDINGLY) to round off the sentence.

Although these conventions are very simple, they cover quite a range of "natural" expressions. The form (1) goes beyond allowing the user a free choice of one word verbs, which often would be insufficient. A verb can be qualified immediately, or at the end of a sentence adverbially or in other ways. Examples of such invoking statements are:

SEND A MEMBERSHIP CARD TO "GRENDEL JONES" AT "THE BEACH HUT".

SEND AN OVERDUE REMINDER TO "DR. FAUST" AT "SANS'SOUCI/BRILLIG DRIVE".

ANNOUNCE "THE CHASED AND THE PURSUER" TO "THE TRASH CAN/75 DREARY LANE" IN SUGGESTIVE TERMS.

ANNOUNCE "THE ILLIBERAL LIBERATION" TO "THE PAPER BACK EGG-HEAD/93 FARM YARD" IN INTELLECTUAL UNDERTONES.

which would invoke four separate subprocedures, characterized by the frameworks left by omitting the quotations.

The proposed conventions also allow free use of prepositions (except THEN, which is best left out) in the invoking framework which is a great aid to naturalness. By allowing the generic name of the elements of a list as an argument, the definition of superlative adjectives (as in THE SHORTEST ITEM) is introduced.

The form (1) invoking statement suggested above can be used when a subprocedure does not create any entities for which names do not yet exist in the invoking procedure. Form (2) is useful when one new name must be introduced, and form (3) when several are needed.

A simple convention for heading subprocedures is to begin with a statement of the form PROCEDURE TO followed by the invoking skeleton in which dummy arguments are embedded. These arguments then can be listed in a statement that begins THE ARGUMENTS ARE. (or THE ARGUMENT IS...). Function subprocedures can be headed PROCEDURE FOR..ING.. where..ING denotes an arbitrary participle (e.g., FORMING, FINDING) and the further dots stand for the function expression with embedded dummy arguments. Some further provisions also will be made for input output, and for user defined conditions.

Introducing the indicative

SNAP, as it has been described so far, consists almost entirely of imperative mood statements. The extended language that is now planned will also include several kinds of indicative mood statements, that will allow statements such as:

(i) THE NAMES OF STRINGS INCLUDE THE SURNAME, THE PRENAME, AND THE ADDRESS.

(ii) THE PRESIDENTIAL RECORD CONTAINS THE SURNAME, THE GIVEN NAME, AND THE DATE OF BIRTH; IN CHARACTER POSITIONS 3 TO 12, 13 TO 25, AND 26 TO 33, RESPECTIVELY


(iv) THE OBJECTS INCLUDE THE PRESIDENT, AND THE VICE-PRESIDENT.

(v) THE PRESIDENT IS DESCRIBED BY THE PRESIDENTIAL RECORD.

(vi) THE VICE-PRESIDENT IS DESCRIBED BY THE VICE-PRESIDENTIAL RECORD.
(vii) THE PRESIDENT IS DESCRIBED BY A BIOGRAPHICAL RECORD.
(viii) THE VICE-PRESIDENT IS DESCRIBED BY A BIOGRAPHICAL RECORD.
(ix) A BIOGRAPHICAL RECORD CONTAINS THE SURNAME, THE GIVEN NAME, AND THE DATE OF BIRTH; IN CHARACTER POSITIONS 3 TO 12, 13 TO 25, AND 26 TO 33, RESPECTIVELY.

The sentences (i)-(iii) are, in effect, verbalized forms of type declaration, format statement and equivalence statement. Sentence (iv) is a type declaration that makes THE PRESIDENT and THE VICE-PRESIDENT the names of objects, which sentences (v) and (vi) relate (by a convention that governs the use of IS DESCRIBED BY) to two strings called THE PRESIDENTIAL RECORD and THE VICE-PRESIDENTIAL RECORD. Statements analogous to (ii) could then be used to describe the internal structure of these strings. Statements (vii) and (viii) take a slightly different tack, using a generic string name. By convention these statements would permit the expressions THE BIOGRAPHICAL RECORD OF THE PRESIDENT and THE BIOGRAPHICAL RECORD OF THE VICE-PRESIDENT to be used as string names, for example in input statements; and in conjunction with sentence (ix) would propagate this association, to allow the use of expressions such as THE SURNAME OF THE PRESIDENT to be interpreted correctly. Further simple sentences, that contain the verb HAS, can be used within the framework of some more simple conventions to introduce the names of objects that are attributes of other objects, and described by strings that thereby become indirect attributes of the latter objects.

Syntactic definitions are of considerable importance, and in this regard the kind of meta-syntactic language, and method of representing the result of a syntactic analysis that were developed in the author's laboratory at M.I.T. some years ago seem a useful basis for further work.

Many further kinds of definitional device can be postulated that seem potentially useful. For example, class inclusional schemes are given an added dimension by the simple tactic which is illustrated by the following sequence of statements, that relate to a file concerning animals in a zoo.

IN THIS PROCEDURE:
THE KINDS OF CATEGORY INCLUDE SUB-KINGDOMS, ORDERS, CLASSES (SINGULAR-CLASS), FAMILIES (SINGULAR-FAMILY), AND SPECIES (SINGULAR-SPECIES).

THE KINDS OF OBJECT INCLUDE ANIMALS.
THE SUB-KINGDOMS OF ANIMALS ARE VERTEBRATES, AND INVERTEBRATES.
THE ORDERS OF VERTEBRATES ARE MAMMALS, BIRDS, REPTILES, AMPHIBIA (SINGULAR-AMPHIBIAN), AND FISH.

THE SPECIES OF APES ARE GORILLAS, CHIMPANZEEZ, AND ORANG-UTANS.

AN ANIMAL IS DESCRIBED BY A RESIDENT RECORD.
A RESIDENT RECORD CONTAINS THE SPECIES, THE DATE OF ACQUISITION, ...

(LOOP START) READ A RESIDENT RECORD. IF THE ANIMAL IS A VERTEBRATE PRINT THE ORDER ...

The example has, amongst other things, an element of metonymy. The word SPECIES appears as a kind of category, that includes GORILLAS, CHIMPANZEEZ etc., and also as the name of a substring of a RESIDENT RECORD. These two uses will be associated, so that when necessary, the contents of the SPECIES field of a RESIDENT RECORD may be compared with the instances of SPECIES in the class inclusional statements. This will make it possible to use the latter words in the data, and to interpret statements such as the IF statement that ends the excerpt.

ACKNOWLEDGMENTS

The work reported in this paper was done in the Graphic Systems Applied Research Laboratory, RCA Laboratories, Princeton, N.J., with which the authors were associated.

REFERENCES

1 Annual reviews of information science
John Wiley and Son 1968
2 M P BARNETT W M RUHSAM
A natural language programming system for mechanical text processing
IEEE Transactions on Engineering Writing and Speech Vol EWS-11 No 2 August 1968 45
3 M P BARNETT
Computer programming in English
Harcourt Brace and World New York Spring 1969
4 W M RUHSAM  M P BARNETT
To be published
5 M P BARNETT  R P FUTRELLE
*Syntactic analysis by digital computer*

CACM Vol 5 1962 515
6 M P BARNETT  M J BAILEY
The shadow V system
Unpublished work