tional Weather Service, Central Region
Computer Programs and Problems
NWS CRCP - No. 3

A SUBROUTINE TO FIND AND EXTRACT DATA FROM AN AFOS DATABASE PRODUCT

Robert L. Somrek
National Weather Service Forecast Office
Chicago, IL

May 1982

noaa NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION / National Weather Service

NOAA, National Weather Service, Central Region
Computer Programs and Problems

NWS - CRCP - No. 3

A SUBROUTINE TO FIND AND EXTRACT DATA FROM AN AFOS DATABASE PRODUCT

Robert L. Somrek
National Weather Service Forecast Office
Chicago, IL

May 1982

UNITED STATES
DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary /
National Oceanic and
Atmospheric Administration
John V. Byrne, Administrator /
National Weather
Service
Richard E. Hallgren, Director

A SUBROUTINE TO FIND AND EXTRACT DATA FROM AN AFOS DATABASE PRODUCT

Robert L. Somrek
National Weather Service Forecast Office
Chicago, IL

May 1982

CONTENTS

A SUBROUTINE TO FIND AND EXTRACT DATA FROM AN AFOS DATABASE PRODUCT

Robert L. Somrek
National Weather Service Forecast Office
Chicago, IL

I. GENERAL INFORMATION

Summary

Many AFOS applications programs require the input of data which are available in an AFOS database product. Given the KSCRF and RDBKF routines callable from UTIL.LB, the retrieval of the AFOS product containing the data is simple. The varying position of the data within similar products and the often involved process of finding and extracting the data, however, have forced many programmers to cumbersome methods of data entry.

A subroutine consisting of routines FIND and XTRACT has been written to aid in searching for and extracting data from AFOS database products. The search routine, FIND, will locate the occurrence and position of a user-defined pattern in the AFOS product. The implementation of metacharacters to alter the normal search process makes the routine very flexible. The extraction routine, XTRACT, will retrieve a copy of the data as well as locate it in the AFOS product.

Evironment

The subroutine runs within the environemnt of the calling program. It is designed to be called from a program running in the background partition of the AFOS Eclipse S/230 minicomputer.

The routines FIND and XTRACT are written in Data General's assembly language for the Eclipse. The routines have been combined into a single subroutine with two entry points.

References

A discussion of the concepts and techniques that can be used in searching for patterns in text is given in

Kernighan, B. and P. Plauger, 1976: Software Tools, Addison-Wesley Publishing Co., 338pp.

The use of text searching routines and metacharacters to enchance the routines can be gleaned from Data General's Text Editor and SUPEREDIT User's Manual.

II. APPLICATION

Program Formulation

The Problem--Finding Patterns in Text: A number of AFOS applications programs have been written to aid the forecaster in analyzing and manipulating the large amount of data available to him. The usefulness of the programs lies in the speed with which data can be handled in complex and lengthy algorithms. While considerable time savings have been achieved in data analysis and manipulation, all too often the task of data entry to the program remains combersome and time-consuming.

Suppose we wish to determine the temperature at twenty stations as input to an applications program. How do we do it?

One method of data entry consists of collecting twenty aviation observations, noting the temperature from each observation, and then manually inputting the data through some means such as an AFOS preformat. Such a method is time-consuming, cumbersome, and filled with the possibility for error.

A better approach to the problem is to incorporate a simple routine into the applications program which can find and read temperatures from aviation observations. While an improvement, this method is so specialized that it has no use outside the specific program that it was written for; something like it will have to be written for a program that requires any other data from an aviation observation as input. Furthermore, this approach often makes no provision for the possibility that the data is not where it is "supposed" to be but may be displaced several character positions.

Obtaining temperatures from aviation observations is a special case of a general problem -- finding patterns in text. What is needed is a routine which can search text (AFOS database product) for a pattern (data). The pattern can be the literal portion of the text that is desired (e.g., determine if a SELS WIBIS is for tornados or severe thunderstorms by searching the WIBIS text for the pattern TORNADO) or a generic description of the text to be searched for (e.g., determine the wind from an aviation observation by searching for the pattern defined by the sequence slash-optional letter E-four numeric characters). Such a routine should be general enough to find almost any data needed for entry to an applications program.

The Solution--A Simple Pattern Finder: The concepts involved in designing a simple pattern finding routine are straight-forward. Given a string of text, a pattern will be said to be found in the text if it contains the same sequence of characters as the pattern. To find the pattern it will be necessary to compare the pattern to the text on a character by character basis.

The concept of the simple pattern finder above suggests that the routine can be separated into three logical tasks. One task will define the position in the text at which to begin searching for the pattern. A second task will perform a similar function by defining the position in the pattern at which to make a comparison with the text character. The third task will make the actual comparison between the text and pattern on a character-by-character basis. Assigning such a modular structure to the routine greatly simplifies the complex job of proceeding through the text and pattern separately but in proper synchronization. It also allows the process of comparing text and pattern characters to be separated from determining what text/pattern match to try next.

Conceptually, the modular pattern finder can be envisioned to operate in the following manner. Upon entry to the pattern finding routine the first module will establish the position in the text at which to begin searching. The second module is then entered to initialize the position in the pattern at which to attempt matching. The third module will then test for identical characters at the current text and pattern positions. Successful matching of the text and pattern characters will force adjustment of the text position to the next character before returning to module 2 for adjustment of the pattern position and reentrance to module 3 for another match test with the new text and pattern characters. The match-then-adjust sequence between modules 3 and 2 will continue until the end of the pattern is reached successfully or until a mismatch occurrs. If the matching process in module 3 indicates dissimilar text and pattern characters, the routine returns to the first module for adjustment of the text beginning position, then to module 2 for initialization of the pattern position, and then again to module 3 where matching is attempted with the new text and pattern characters.

Figure 1 illustrates the process of searching for the pattern ON in the text SOONER. The pointers represent the functions performed in modules 1 and 2 of defining the text and pattern characters to be tested in module 3.

x x x x x x x x x x

```
SOONER          ON          --Module 1 defines text position to begin search at S; passes
↑                             to module 2

SOONER          ON          --Module 2 defines pattern position ot begin search at O
↑               ↑

SOONER          ON          --Module 3 tests S against O. Finding dissimilar characters
↑               ↑             flow passes to module 1 where new text position is defined
                              at first O
SOONER          ON          --Module 2 redefines pattern position at O
 ↑              ↑

SOONER          ON          --Module 3 finds a match and adjusts text postion to second O
  ↑             ↑

SOONER          ON          --Module 2 adjusts pattern position to next character at N
  ↑              ↑

SOONER          ON          --Module 3 attempts a match and fails. Flow passes to module 1
  ↑                           where beginning text position is redefined at second O

SOONER          ON          --Module 2 defines pattern position be begin search at O
  ↑             ↑

SOONER          ON          --Module 3 finds a match and adjusts text position to N
   ↑            ↑

SOONER          ON          --Module 2 adjusts pattern position to next character at N;
   ↑             ↑             Module 3 finds a match--pattern has been found in text!
```

OPERATION OF A SIMPLE PATTERN FINDER

Figure 1

A Better Solution--The Introduction of Metacharacters: While a routine to search text for a pattern consisting of a literal string of characters is useful, it quickly becomes apparent that such a pattern finder is restricted in what it can find. How does one search for such patterns as WWUS or SRUS25 or SDUS8 which are generically similar yet cannot be found by any single literal pattern?

The introduction of metacharacters -- characters whose meaning have been preempted to represent special patterns -- provides a simple pattern finding routine with the flexibility to search for patterns which cannot be expressed literally. Metacharacters will allow the pattern finder to search text for patterns such as those which consist of classes of characters or which match only at a particular position on a line or which are of indefinite length.

While a list of useful metacharacters might be extremely long, a simple pattern finder can be enhanced considerably by including metacharacters to represent the following special classes of patterns:

--a pattern metacharacter (ANY) that will match any current text characacter; the pattern A*C matches such varied text as ABC, A2C, A+C, and even A*C (in examples of the use of metacharaters, an asterisk denotes the metacharacter; specific characters will be assigned in the software implementation);

--a metacharacter (ANYA) that matches any alphabetic text character such that A*C matches ABC, or AZC but not A2C or A+C;

--a metacharacter (ANYN) that will only match a single numeric character; A*C will match A2C or A7C but not ABC, A+C, or A27C;

--a metacharacter (NCHAR) to match any character but that following the metacharater so that a pattern like A*BC will match AAC, AZC, or A+C but not ABC;

--a metacharacter (OCTAL) that will allow a match with a text character whose octal value is delimited by the metacharacter; the metacharacter will allow matching the full ASCII character set; *nnn* will match a text character whose octal value is nnn;

--a metacharacter (BOL) to allow a match with the text only if the pattern following the metacharacter occurs at the beginning of a line; *ABC is a pattern that matches the text ABC only if it occurs as the first three characters on a line;

--a metacharacter (CHRCL) that will signal that any of the literal characters in the group to follow match the current text character; the pattern A*BCD*E will match ABE, ACE, or ADE but no other text string (note the paired use of the metacharacter to delimit the group of characters);

--a special metacharacter (CLOSR) that will allow matching zero or more occurrences of the next pattern character in the text string; consider the pattern A*BC which matches AC, ABC, ABBC, etc.; the "closure" metacharacter may preceed any of the metacharacters described above;

--a metacharacter (OFF) to "turn off" the special meaning of any of the metacharacters described above (including itself) and allow the pattern to be matched literally against the current text character; A**B will match A*B.

The introduction of all the metacharacters but CLOSR add only minor difficulty to the design of the pattern finder. They are best absorbed in matching module 3 where their special meaning can be checked against the current text character.

Closure matching causes some difficulty. Encountering a closure match should cause a loop on the character to be tested thereby finding as many occurrences of the character as possible until matching fails. Searching should then continue in the normal fashion from the point of the closure failure. But what if the remainder of the pattern fails? It is not necessarily an indication that the pattern cannot be found in the text but perhaps only that the closure match was too long. Consider the pattern *AA in this example: the pattern will match the text AA only if the closure portion of the pattern (*A) is limited to matching the first A in the text. The implication in this example is that for every instance that the pattern following a closure fails, the routine must go back and shorten the number of matches made by the closure pattern by one and then retry matching the remainder of the pattern.

The backtracking described above is the simplest case of a search failure due to incorrect closure matching. If the pattern is composed of several closure entries, it is necessary to systematically backtrack through all previous closures to correctly determine the routine's success or failure. The easiest way to handle such a backtracking scheme is through a recursive procedure. Closure matching will be implemented in module 2 -- the pattern position module.

Software Implementation and Structure

The choice of software language is important in implementing the pattern finder in line with its conceptual form. Inherent in the design outlined is the use of pointers to establish and synchronize the progression through text and pattern match positions (graphically illustrated in Figure 1). Additionally, the implementation of the CLOSR metacharacter through a recursive technique requires the allocation of storage in the run-time environment. Since FORTRAN limits the easy use of such design concepts, assembly language was chosen for the software implementation of the pattern finder. Assembly language also helps minimize the amount of memory and time required for the routine's execution.

The tri-modular structure discussed in the design of the pattern finder is easily carried to the software implementation (see the annotated listing in Section IV). The first module, TXTPOS, establishes the position in the text to begin searching for the pattern. The second module, PATPOS, defines the position in the pattern at which to make a comparison with the text character. The third module, MATCH, performs the comparison between text and pattern characters.

Upon entry, the subroutine notes the entry point and quickly passes to the first of the three modules, TXTPOS. TXTPOS establishes the necessary linkage to the calling routine and places the addresses of the variables in the argument list on the subroutine's stack. For convienence, a byte pointer (NEDLE) to the first match set in the pattern (PATRN) is initially created here rather than in PATPOS. TXTPOS then creates a byte pointer (CURSR) to the first element in the array (TEXT) holding the text to be searched. The pointer is offset to a user-prescribed location (START) within the text array at which to begin searching. An additional byte pointer (EOS) is created to indicate the last location (END) at which to attempt to find the pattern. Having created the necessary pointers, the module tests for the end of text (ETX character from the AFOS product) and then passes to the second module, PATPOS.

PATPOS establishes the current pattern character position (NEDLE) and then determines whether or not a closure match is specified. (The implementation of closure matching in this module is discussed below.) If closure matching is not indicated, program flow passes quickly to the third module, MATCH.

MATCH tests the current text and pattern characters for similarity. Before proceeding to the test, however, a check is made to see if the pattern character is one of the special metacharacters. (Figure 2 describes the characters used in implementing each metacharacter.) MATCH then compares the text and pattern characters using a routine (denoted CHAR, NCHAR, ANY, etc.) designed for the type of matching specified. At this point program flow is dependent on the result of the matching process. Character testing in the MATCH module can produce any of three results:

1. Successful matching of the text and pattern characters;
2. Unsuccessful matching as a result of different text and pattern characters;
3. Unsuccessful matching because all the text has been checked and no match of the text and pattern

is possible.

With successful matching of text and pattern characters in the MATCH module, the CURSR byte pointer is advanced to the next text character to be matched before a return is made to PATPOS for adjustment of the NEDLE pointer and reentry to MATCH. The interaction between MATCH and PATPOS will continue in this manner as long as text and pattern character are the same. When the entire PATRN has been found successfully in TEXT, return to TXTPOS allows updating the argument list for a return to the calling program. If unsuccessful matching in MATCH is the result of different text and pattern characters, return is made to TXTPOS for adjustment of the CURSR pointer to the next character in TEXT and a re-initiation of the entire "find" process. Unsuccessful matching which indicates that the pattern does not exist in the text will force a return to TXTPOS for generation of an error code and return to the calling program.

Program flow in PATPOS on return from MATCH and in TXTPOS on return from PATPOS is dependent on what happened in MATCH and PATPOS before the return was taken. As in MATCH the branching of program flow is a function of the outcome of the match of the current pattern and text characters. For this reason the concept of sub-modules implicit in MATCH has been extended to both TXTPOS and PATPOS. The sub-modules have been given the generic names SUCES (action taken when pattern and text characters have been matched successfully), TFAIL (action resulting from a total failure to find the pattern in the text), and CFAIL (action taken due to a failure to find the current pattern character at the current text position). Each sub-module performs the necessary pointer and variable adjustments necessary to continue with the next logical portion of the subroutine.

Closure matching has been implemented in the PATPOS module through the use of a recursive procedure. The recursion makes use of the Eclipse's push-down stack to allow easy storage and protection of the state of the subroutine variables before redefining then through reentrant use of the PATPOS code. The dynamic nature of the stack allows repeated recursive entry to PATPOS without complicated storage allocation procedures.

With the implementation of closure matching in mind, we can take a second look at program flow through PATPOS. Entry to the module initiates a call to the FORTRAN library routine FSAV. FSAV not only saves the current state of the calling routine (either TXTPOS or PATPOS itself) but also allocates space on the run-time stack for a new stack frame for PATPOS. The frame contains a nine word header (see the FORTRAN IV Run-Time Library User's Manual for details) and eight words of user storage to keep track of the bookkeeping details associated with a recursive entry to the module. PATPOS then determines whether closure matching is specified at the current pattern position. If closure matching is indicated, the CURSR pointer is adjusted to the character to be matched and program flow passes to the MATCH module. With each successful match of the closure character, the number of characters in the text which make up the match is incremented and stored on the frame for later use.

When a match of the closure pattern character finally (or initially) fails, the CURSR and NEDLE pointers are stored on the frame. The "find" process then attempts to match the remainder of the pattern with a recursive call to PATPOS. When subsequent non-closure matching fails, the end point of the most recent closure match is returned to; the length of the closure match is shortened by one; and another attempt is made to match the remainder of the pattern. At some point the number of matches in the closure will have been reduced to zero and a match of the pattern is still not possible; return is then made to a CFAIL sub-module and the subroutine attempts to find PATRN elsewhere in TEXT.

If closure matching fails to find the pattern, return is made to the TFAIL sub-module of TXTPOS before returning to the calling program.

The text extraction subroutine XTRACT has been combined with FIND and is implemented as a second entry point to the subroutine. Once entry to XTRACT is flagged, the routine jumps to FIND and proceeds as described above.

Upon successfully finding the pattern in the text and determining that entry was through XTRACT, program flow passes to a short routine which transfers the text to an output array. With the transfer complete, return is made to the calling program.

Machine Requirements

The subroutine runs within the enviroment of the calling program. It is designed to be called from a program running in the background partition of the AFOS Eclipse S/230 minicomputer.

The relocatable binary file requires 860 bytes of disk storage. When loaded to an RDOS save file, the subroutine requires 239 words of memory for execution. A minimum of 35 words is required on the run-time stack; additionally, each recursive entry to the PATPOS module rquires another 17 words on the stack.

The subroutine is supported by the FORTRAN IV library routines .CPYL, FSAV, and FRET.

Typical execution times to find and extract a five character literal pattern in text three AFOS blocks in length are less than 0.75 second. Although execution times using patterns with metacharacters are also generally less than one second, some care should be taken in constructing patterns with the metachaacters. Using a bad pattern such as *?x (closure-any character-literal x), where the character x does not exist in the three blocks, can increase the execution time to more than a minute.

* * *

III. PROCEDURE

Assembling and Loading the Subroutine

With the source code for the subroutine in file FIND.SR, the routine is assembled using the CLI command MAC FIND.SR. Output from the macroassembler will be found in the relocatable binary file FIND.RB.

To load the binary file to an RDOS save file, use the CLI command

```
RLDR mainprogram FIND subroutines user-libraries FORT.LB.
```

Calling the Subroutine

When used with a FORTRAN program, execution of the subroutine is initiated by a CALL statement. The form of the calls are

```
CALL FIND (TEXT,PATRN,START,END,EROR)
or
CALL XTRACT (TEXT,PATRN,START,END,EROR,XTEXT)
```

On entry to the subroutine, the argument list is defined as follows:

XTEXT--the name of a packed integer array holding the text to be searched
PATRN--the name of a packed integer array or a literal string which holds the pattern to search for. If PATRN is passed to the subroutine in an array, the pattern must be terminated with a null byte.
START--an integer variable which defines the character position (not array element number) in TEXT at which to start searching for the pattern. Good use of this argument will allow the searching process to "skip over" a leading part of text which is known not to contain a match for the pattern.
END----an integer variable which defines the character position (not array element number) in TEXT a which to end searching for the pattern. Good use of this argument will stop the search process when it is known that a match for the pattern will not be found past this point in the text. For AFOS database products, searching will automatically terminate at the ETX character if END exceeds the length of the product.
EROR---an integer variable to hold an error code returned by the subroutine.
XTEXT--the name of an integer array to hold the text extracted by the subroutine.

Examples of calls to the subroutine are: CALL FIND (ITXT,IPAT,ISTRT,IEND,IER) and
CALL XTRACT (IAR,"PATTERN",ISTRT,IEND,IER,IIAR)

The results of the subroutine's execution are returned through the argument list. On return, the argument list contains:

TEXT---the text array is left unchanged
PATRN--the pattern array or literal string is left unchanged
START--upon successful completion of the subroutine the variable is updated to contain the character position at which the first element of the pattern was found in the text. For AFOS database products, remember that the number will include the 20 bytes in the AFOS block zero header.
END----if the pattern is found the variable is updated to contain the character position

immediately following the matched text. END - START = length of match.

ERDR---the variable is updated to hold the error code returned by the subroutine. Successful completion (pattern found in text) returns a +1. If the pattern cannot be found in the text, a -1 is returned.

XTEXT--the array will contain a copy of the text which matches the pattern. The text will be in packed format and will begin in the left byte of the first array element. A terminator to mark the end of the text is not included.

Use of the Metacharacters

All the metacharacter described in the program formulation above are implemented in the subroutine. Characters which should have a low frequency of occurrence in AFOS products were chosen. Figure 2 lists the generic name of each metacharacter along with the character implemented in the subroutine, a description of function, and an example of its use in a pattern.

The Appendix contains a FORTRAN program which can be used to examine the features of the subroutine.

Cautions and Restrictions

Since the subroutine receives all its input data through the argument list, some care must be taken to insure that the list accurately reflects the user's intentions and that it conforms to a few conventions assumed by the subroutine.

....The PATRN array must be terminated by a null byte. The null byte serves as an end-of-pattern character. When passing PATRN as a literal string, the FORTRAN compiler will provide the terminating null byte.

....Both arrays TEXT and PATRN must be in packed (i.e., two characters per computer word) format. This restriction allows the subroutine to operate most efficiently and is a requirement resulting from the caution above.

....Both START and END must be variables containing the character number in TEXT at which to begin and end searching and not TEXT array element numbers.

....When using the subroutine with a non-AFOS product, the user must be sure the value of END is within the text being searched.

....No check is made to see if XTEXT has been DIMENSIONed large enough to hold the entire output of the subroutine.

....An AFOS end-of-line is delimited by the character pair carriage return/line feed. Searching for text across the end of a line must include a CR/LF pair at the appropriate place in PATRN. The CR/LF pair may be included in PATRN using the OCTAL metacharacter (CR LF = <015><012>).

....Since the BOL metacharacter implies a leading carriage return or carriage return/line feed combination, the pattern BOL x, where x is the first character in TEXT, will not be found by the routine.

x x x

NAME	CHARACTER	FUNCTION/EXAMPLE
ANY	?	Accept any character at the current text position as a match with the pattern Example: A?C will match ABC, A2C, A+C, A?C, etc.
ANYA	&	Accept any (upper case) alphabetic character as a match Example: A&C will match text like AAC thru AZC but not text like A1C or A+C
ANYN	#	Accept any single numeric character at the current text position as a match Example: A#C will match A1C thru A9C but not text like ABC, A+C or A12C
NCHAR	!	Accept any character as a match with the pattern except the character following this metacharacter Example: !ABC will match all text of three characters ending in BC except ABC
OCTAL	< >	Accept the ASCII text character whose octal value is the same as the number between the angle brackets as a match with this pattern Example: C<101>R will only match the text CAR
BOL	%	Accept the next text character as a match only if it occurs at the beginning of a line Example: %FIRST will match the text FIRST only is it occurs as the first five characters on a line
CHRCL	$	Accept any of the characters enclosed in the paired dollar signs as a match with the current text character. All metacharacters but OFF are interpreted literally within CHRCL. To include a dollar sign in CHRCL use the construct @$. Example: H$AIOU$T will match HAT, HIT, HOT or HUT
CLOSR	*	Accept as many of the character following * at this point in the text as a match with the pattern. This metacharacter can be used with any of the seven metacharacters described above. Example: M*ET will match MT, MET, MEET, etc.; 19*# will match any year century
OFF	@	Interpret the next character in the pattern literally and not as a meta-character Example: 50@% will match 50%; @@ will match @

DESCRIPTION OF THE METACHARACTERS

Figure 2

IV. PROGRAM LISTING

```
        .TITL FIND          ;A SUBROUTINE TO FIND AND EXTRACT DATA FROM AN AFOS DATABASE PRODUCT
         .REV 2,0           ;  by Robert L. Somrek, WSFO Chicago
        .ENT FIND,XTRACT
        .EXTN FSAV,FRET
        .EXTD .CPYL

        .ZREL
.PATPOS:PATPOS-2

        .NREL
        TEXT=-167           ;Frame set-up:  TEXT--text string or array to be searched
        PATRN=TEXT+1        ;               PATRN--pattern string or array to search for
        START=PATRN+1       ;               START--character position in TEXT at which to start search
        END=START+1         ;               END--character postion in TEXT at which to end search
        EROR=END+1          ;               EROR--error code to be returned on completion of subroutine
        XTEXT=EROR+1        ;               XTEXT--output array
        NEDLE=XTEXT+1       ;               NEDLE--byte pointer to character in PATRN
        CURSR=NEDLE+1       ;               CURSR--byte pointer to character in TEXT
        .FS=CURSR-TEXT+1    ;Frame size defined for six arguments and two stack variables


FIND:   PSHJ   TXTPOS       ;Save point of entry on stack then go to TXTPOS
        FRET                ;On return from modules, return to calling routine
```

x x x

;MODULE TXTPOS -- defines position in TEXT at which to begin searching for PATRN

```
          .FS
TXTPOS:JSR     @.CPYL         ;Copy argument list to subroutine's frame

       LDA     0,PATRN,3      ;Create byte pointer to first element
       MOVZL   0,0            ; in PATRN and
       STA     0,NEDLE,3      ; store in on frame
       LDA     2,TEXT,3       ;Create byte pointer to first element
       MOVZL   2,2            ; in TEXT
       LDA     1,@END,3       ;Create position pointer to last character
       ADD     2,1            ; in TEXT at which to try matching and
       STA     1,EOS          ; store it
       LDA     1,@START,3     ;Offset TEXT byte pointer to START position
       ADD     1,2            ; in TEXT array then
       SBI     1,2            ; adjust pointer for displacement to first element;
       STA     2,CURSR,3      ; store it on frame
       LDA     1,EOS          ;Check if START was specified
       SUBZ#   1,2,SZC        ; past END
       JMP     TTFAIL+1       ; If it was, return to calling program with EROR=-1; otherwise continue

NXTTXT:LDB     2,3            ;Load TEXT byte and
       LDA     1,ETX          ; end-of-transmission character then
       SUB#    1,3,SNR        ; test TEXT byte against it.  Is this end of input string?
       JMP     TTFAIL         ; Yes - PATRN not found in TEXT
       JSR     @.PATPOS       ; No - test character
       JMP     TCFAIL         ;Return from PATPOS:  Unsuccessful match of character
       JMP     TSUCES         ;                     Successful match of entire PATRN
       JMP     TTFAIL         ;                     Failure to find PATRN in TEXT
```

module continued on next page...

```
TSUCES:LDA     3,16           ;Restore TXTPOS's frame pointer
       LDA     0,CURSR,3      ;Load beginning CURSR postion (ending position returned in AC1); then
       SUB     0,1            ; Ending position - CURSR = Length of match
       LDA     2,@START,3     ;Load START position and
       ADD     1,2            ; add length of match to compute
       STA     2,@END,3       ; end position of match; store it on frame
       SUBZL   2,2            ;Generate +1 for successful completion of FIND and
       STA     2,@EROR,3      ; store it on frame
       POPJ                   ;Return to entry point

TCFAIL:LDA     3,16           ;Restore TXTPOS's frame pointer
       LDA     0,NEDLE,3      ;Reset NEDLE to first element of PATRN
       ISZ     CURSR,3        ;Bump CURSR to look at next character in TEXT
       LDA     2,CURSR,3      ;Load new CURSR and
       LDA     1,EOS          ; end-of-string position pointer;
       SUBZ#   1,2,SZC        ; is new CURSR past last position to try matching?
       JMP     TTFAIL         ;  Yes - PATRN not found in TEXT
       ISZ     @START,3       ;  No - increment match START character position and
       JMP     NXTTXT         ;      try matching next character in TEXT

TTFAIL:LDA     3,16           ;Restore TXTPOS's frame pointer
       ADC     0,0            ;Generate -1 for unsuccessful completion of FIND and
       STA     0,@EROR,3      ; store it on frame
       FRET                   ;Return to calling routine

;Constants and temporary storage

ETX:     203            ;ASCII end-of-text character (with bit 0 set)

;End of TXTPOS
```

x x x

```
;MODULE PATPOS -- defines position in PATRN at which to make comparison with TEXT character. PATPOS also
                 implements recursive CLOSURE matching.


      FOSP=-177          ;Frame set-up:  FOSP--previous frame's pointer (stored in this frame's header)
      FRTN=-170          ;               FRTN--return address
      FAC1=-172          ;               FAC1--storage for AC1
      CLZRCT=-162        ;               CLZRCT--counter for taimes a closure on this frame matches

      FSAV               ;Create new frame with 9 word header and
      10                 ; 8 word storage

PATPOS:LDA    0,NEDLE,2  ;Load value of NEDLE and
      LDA    2,CURSR,2   ; CURSR from previous frame
      SUBZL  1,1         ;Generate +1 for closure match count and
      STA    1,CLZRCT,3  ; store it on this frame

NXTPAT:LDB    0,1        ;Load PATRN byte
      MOV    1,1,SNR     ;Is this end of PATRN?
      JMP    FOUND       ; Yes - PATRN found in TEXT
      LDA    3,CLOSR     ; No - load metacharacter for CLOSURE and
      SUB#   1,3,SZR     ;      test whether PATRN byte is a CLOSURE entry
      JMP    NCLOZR      ;Not a CLOSURE entry - go to NCLOZR

CLOZUR:INC    0,0        ;A CLOSURE entry - bump NEDLE to character to be matched
      LDA    3,16        ;Restore this frame's pointer to use in
      STA    0,NEDLE,3   ; storing beginning position of this match set for repeated
      JSR    MATCH       ; entry to MATCH
      JMP    CCFAIL      ;Return from MATCH:  Unsuccessful match of closure character
      JMP    CSUCES      ;                    Successful match of closure character
      JMP    CTFAIL      ;                    Failure to find PATRN in TEXT

      CSUCES:LDA    3,16        ;Successful match of closure character - restore frame pointer to use in
            LDA    0,NEDLE,3    ; returning NEDLE to beginning match set and in
            ISZ    CLZRCT,3     ; incrementing number of matches in this closure
            JMP    .-7          ;Now try matching next TEXT byte with this closure character
      CCFAIL:INC    0,0         ;Unsuccessful match of closure character - bump NEDLE to next match set
            LDA    3,16         ;Restore this frame's pointer to use in
            STA    0,NEDLE,3    ; storing current NEDLE and
            STA    2,CURSR,3    ; current CURSR for use on return from recursive call to PATPOS
       RECUR:JSR    @.PATPOS    ;Now test the remainder of PATRN (RECURSIVE CALL)
            JMP    RCFAIL       ;Return from recursive PATPOS:  Unsuccessful match of next character
            JMP    RSUCES       ;                               Successful match of remainder of PATRN
            JMP    RTFAIL       ;                               Failure to find PATRN in TEXT
```

module continued on next page...

```
        RSUCES:JMP   FOUND+2       ;Successful match of rest of PATRN - return to SUCES submodule
        RCFAIL:DSZ   CURSR,3       ;Unsuccesful match of next character - bump CURSR back one and
               DSZ   CLZRCT,3      ; reduce number of matches in this closure by one; then
               JMP   RECUR         ; try again to match remainder of PATRN
               FRET                ;Next character doen't match 0 closure matches - return to CFAIL
        RTFAIL:JMP   NTFAIL+1      ;Failure to find PATRN in TEXT; return to TFAIL submodule

    CTFAIL:INC    0,0         ;Failure to find PATRN in TEXT - but first bump NEDLE to next match set;
           LDB    0,1         ; load PATRN byte; and
           MOV    1,1,SNR     ; test for end of PATRN
           JMP    FOUND       ; END of PATRN - successful match since closure ended onn PATRN EOS
           LDA    1,EOS       ; Not end of PATRN - but does closure end
           SUBZ#  1,2,SNC     ;  on TEXT EOS
           JMP    NTFAIL      ;  No - return to TFAIL submodule on previous frame
           SBI    1,2         ;  Yes - bump CURSR back to last TEXT character;
           LDA    3,16        ;        reduce closure number
           DSZ    CLZRCT,3    ;        by one; then
           JMP    RECUR-2     ;        go try matching PATRN again

NCLOZR:JSR    MATCH
    JMP    NCFAIL       ;Return from MATCH:  Unsuccessful match of character
    JMP    NSUCES       ;                    Successful match of character
    JMP    NTFAIL       ;                    Failure to find PATRN in TEXT

    NSUCES:INC    0,0         ;Successful match of character - bump NEDLE to next match set and
           JMP    NXTPAT      ; try matching next entry in PATRN

        FOUND: LDA   3,16          ;Restore this frame's pointer
               MOV   2,1           ;CURSR now points 1 character past end of matched PATRN
               LDA   2,FOSP,3      ;Restore previous frame's pointer for use in
               STA   1,FAC1,2      ; returning last CURSR position and
               ISZ   FRTN,2        ; adjusting return address for
               FRET                ; return to SUCES submodule on previous frame

    NCFAIL:FRET               ;Unsuccessful match of character - return to CFAIL submodule
    NTFAIL:LDA    3,16        ;Failure to find PATRN in TEXT - restore this frame's pointer for use in
           LDA    2,FOSP,3    ; retrieving previous frame's pointer; then
           ISZ    FRTN,2      ; adjust return address
           ISZ    FRTN,2      ; for
           FRET               ; return to TFAIL submodule on previous frame

;Constants and temporary storage
   CLOSUR:  52          ;Closure metacharacter:  *

   EOS:     0           ;Byte pointer to last TEXT character to be tested

;End of PATPOS
```

* * *

```
;MODULE MATCH -- compares TEXT to PATRN


MATCH: STA    3,RTNAD        ;Save return address

       LDB    2,3            ;Load TEXT byte and
       LDA    1,ETX          ; end-of-transmission character then
       SUB#   1,2,SNR        ; text TEXT byte against it. Is this end of input string?
       JMP    MTFAIL         ;  Yes - PATRN not found in TEXT
       LDB    0,1            ;  No - load PATRN byte
       DSPA   1,META1        ;Is PATRN byte a special metacharacter?
       DSPA   1,META2

              CHAR:  SUB#   1,3,SNR       ;Does PATRN byte match TEXT byte?
                     JMP    MSUCES        ; Yes
                     JMP    @RTNAD        ; No - return to PATPOS at CFAIL submodule
              ANY:   JMP    MSUCES        ;Any TEXT byte matches this PATRN metacharacter
              ANYA:  CLM    3,3           ;Is TEXT byte an (upper case) alpha character?
                            101           ; (octal value of byte between 101 and
                            132           ; 132)?
                     JMP    @RTNAD        ; No - return to PATPOS at CFAIL submodule
                     JMP    MSUCES        ; Yes
              ANYN:  CLM    3,3           ;Is TEXT byte a numeric character
                            60            ; (octal value of byte between 60 and
                            71            ; 71)?
                     JMP    @RTNAD        ; No - return to PATPOS at CFAIL submodule
                     JMP    MSUCES        ; Yes
              NCHAR: INC    0,0           ;Bump NEDLE past NCHAR metacharacter and
                     LDB    0,1           ; load PATRN byte
                     SUB#   1,3,SZR       ; Is PATRN byte different than TEXT byte?
                     JMP    MSUCES        ; Yes
                     JMP    @RTNAD        ; No - return to PATPOS at CFAIL submodule
              OCTAL: STA    2,TCURSR      ;Temporarily store CURSR then
                     SUBO   2,2           ; clear AC to use in building octal number
                     LDA    3,OCTEND      ;Load OCTAL metacharacter terminator
                     INC    0,0           ;Bump NEDLE past OCTAL metacharacter and
                     LDB    0,1           ; load next byte
                     SUB#   1,3,SNR       ;Is this end of OCTAL?
                     JMP    .+7           ; Yes
                     ADDZL  2,2           ; No - left shift converted number three
                     MOVZL  2,2           ;       bits to make room for next digit
                     ANDI   7,1           ;      Mask 3 rightmost bits and
                     IOR    1,2           ;       add to conversion so far;
                     JMP    .-11          ;       go get next character
                     MOV    2,1           ;Put result in correct AC;
                     LDA    2,TCURSR      ; restore CURSR; and
                     LDB    2,3           ; reload TEXT byte
                     JMP    CHAR          ;Go test octal value against TEXT byte
```

module continued on next page...

```
BOL:    LDA    1,CRRTN      ;Load carriage return as PATRN byte then
        SUB#   1,3,SZR      ; test against TEXT byte. Do they match?
        JMP    @RTNAD       ; No - return to PATPOS at CFAIL submodule
        INC    2,2          ; Yes - bump CURSR to
        LDB    2,3          ;       next character then
        SBI    3,1          ;       load line feed
        SUB#   1,3,SZR      ;       Are they same?
        JMP    MSUCES+1     ;         No - don't bump CURSR
        JMP    MSUCES       ;         Yes - go bump CURSR to next TEXT byte
CHRCL:  INC    0,0          ;Bump NEDLE past character class metacharacter and
        LDB    0,1          ; load character
        LDA    3,OFF.       ;Test for OFF metacharacter (allows a $ or @ in
        SUB#   1,3,SZR      ; CHRCL). Is this OFF?
        JMP    .+3          ; No
        INC    0,0          ; Yes - Bump NEDLE to next
        LDB    0,1          ;       character and
        LDB    2,3          ;       reload TEXT byte
        SUB#   1,3,SNR      ;Does PATRN byte match this character class entry?
        JMP    .+10         ; Yes
        INC    0,0          ; No - bump NEDLE to next class entry and
        LDB    0,1          ;      load it
        LDA    3,CCLEND     ;Is this end
        SUB#   1,3,SNR      ; of character class?
        JMP    @RTNAD       ; Yes - return to PATPOS at CFAIL submodule
        LDB    2,3          ; No - reload TEXT byte and
        JMP    .-10         ;      try matching again
        LDA    3,CCLEND     ;TEXT byte matches character class entry - now find end of CHRCL
        INC    0,0          ;Bump NEDLE to next character and
        LDB    0,1          ; load it
        SUB#   1,3,SNR      ;Is this end of character class?
        JMP    MSUCES       ; Yes
        JMP    .-4          ; No - continue bumping NEDLE to end of character class
OFF:    INC    0,0          ;Bump NEDLE past OFF metacharacter and
        LDB    0,1          ; load PATRN byte then
        JMP    CHAR         ; go to CHAR to test against TEXT byte

MSUCES:INC    2,2          ;PATRN byte matches TEXT byte - bump CURSR to next byte in TEXT
       LDA    1,EOS        ;Load end-of-string position pointer
       SUBZ#  1,2,SZC      ;Is new CURSR past last position to try matching?
MTFAIL:ISZ    RTNAD        ; Yes - adjust return address for return to TFAIL submodule
       ISZ    RTNAD        ; No - adjust return address for return to SUCES submodule
       JMP    @RTNAD       ;Return to PATPOS
```

module continued on next page...

```
;Constants and temporary storage
     RTNAD:   0

              41              ;Dispatch Tables - if PATRN byte is a special metacharacter, program
              46              ; control is passed to one of the special matching routines
     META1:   NCHAR           ;  Character:  !          Function:  match any but next character
              177777
              ANYN            ;              #                     match any single numeric character
              CHRCL           ;              $                     match any character between $'s
              BOL             ;              %                     match only at beginning of line
              ANYA            ;              &                     match any alphabetic character
              74
              100
     META2:   OCTAL           ;              <nnn>                 match character with octal value nnn
              177777
              177777
              ANY             ;              ?                     match any character
              OFF             ;              @                     match next character literally

     OFF.:   100              ;Off metacharacter:  @
     CCLEND: 45               ;Character class metacharacter terminator:  $
     CRRTN:  15               ;ASCII cariage return
     TCURSR: 0                ;Temporary storage for CURSR
     OCTEND: 76               ;Octal metacharacter terminator:   >

;End of MATCH
```

x x x

```
;EXTRACTION ENTRY POINT -- entry point to subroutine when TEXT is to be found and returned to caller


XTRACT:PSHJ   TXTPOS        ;Save point of entry on stack then go to TXTPOS

       LDA    2,XTEXT,3     ;On return from modules, create byte pointer to first element of
       MOVZL  2,2           ; output array
       NEG    1,3           ;Setup number of characters in matched pattern as counter

  COPY:LDB    0,1           ;Load byte from TEXT then
       STB    2,1           ; transfer it to XTEXT
       INC    3,3,SNR       ;Check number of characters remaining to be transferred to XTEXT and
       FRET                 ; return to calling routine if none remain
       INC    0,0           ; otherwise, bump TEXT pointer and
       INC    2,2           ; XTEXT pointer to next character positions
       JMP    COPY          ;Go copy next character

       .END

;End of subroutine
```

x x x

APPENDIX

```
 INTEGER TEXT(321),PATTERN(8),START,FINISH,EROR,XTEXT(7)
 COMMON /FINDR/TEXT
 DATA TEXT/"THE ERUPTION OF MOUNT ST. HELENS, WHICH BEGAN IN A MINOR WAY
+ON MARCH 27, WAS THE FIRST IN THE CONTINENTAL U.S. SINCE THE
+CASCADES LASSEN PEAK, 400 MILES TO THE SOUTH, LAUNCHED A
+THREE-YEAR SIEGE OF ERUPTIONS IN 1914. GEOLOGISTS ESTIMATE
+THAT ST. HELENS SPEWED OUT ABOUT 1.5 CUBIC MILES OF DEBRIS--
+A BLAST ON THE SAME ORDER OF MAGNITUDE AS THE ONE IN A.D. 79
+FROM ITALY'S VESUVIUS THAT COVERED POMPEII AND HERCULANEUM
+WITH ASH AND MUD. THE ERUPTION FELLED 50 SQUARE MILES OF
+TIMBER WORTH MORE THAN $600 MILLION, CAUSED AN ESTIMATED
+$195 MILLION IN DAMAGE TO WHEAT, ALFALFA AND OTHER CROPS
+AS FAR EAST AS MISSOULA AND BURIED 900 MILES OF ROAD UNDER
+ASH."/

 WRITE (10,1) TEXT
1 FORMAT (1X//1X'WELCOME!'//
 + 1X'This is a program written to acquaint you with the use of the FIND/XTACT'/
 + 1X' subroutine.'//
 + 1X'I will print a paragraph of text below that you can use to try the subroutine.'/
 + 1X' When I prompt you for a pattern, enter the pattern you want me to look for. The'/
 + 1X' pattern can be a literal pattern (e.g., ESTIMATE) or can use any of the'/
 + 1X' metacharacters described in the accompanying applications paper (e.g., 19*$).'/
 + 1X' For this demonstration, please limit your pattern to 13 characters.'/
 + 1X' I will also ask you for the beginning and ending character positions between'/
 + 1X' which to search the text.'//
 + 1X'On return from the subroutine, I will first tell you if I found your pattern in'/
 + 1X' the test text.  If I did, I will tell you at what character positions the pattern'/
 + 1X' starts and ends (for this demo there is no carriage return/line feed'/
 + 1X' combination at the end of each line--the % metacharacter will not work).  I will'/
 + 1X' also print a copy of the text that I found to match your pattern.'//
 + 1X'To terminate the program, enter ZZ when I prompt you for a pattern.'//
 + 1X'Ready--here is the text...'//
 + 2(11X30A2/)11X28A2/11X29A2/2(11X30A2/)11X29A2/3(11X28A2/)11X29A2/11X2A2///)
```

appendix continued on next page...

```
2 EROR=0
  DO 3  I=1,7
3 XTEXT(I)='  '

  WRITE (10,4)
4 FORMAT (1X'Enter your pattern:      'Z)
  READ (11,5) PATTERN(1)
5 FORMAT (S14)
  IF (PATTERN(1).EQ.'ZZ')  GOTO 9
  ACCEPT 'At what character position do you want me to begin looking for your pattern?     ',START
  ACCEPT "And at what character position should I stop (don't exced 642)?     ",FINISH
  IF (START.GT.FINISH.OR.FINISH.GT.642)  GOTO 8
  CALL XTRACT (TEXT,PATTERN,START,FINISH,EROR,XTEXT)
  IF (EROR.EQ.-1)  GOTO 7
  WRITE (10,6) START,FINISH,XTEXT
6 FORMAT (1X'I FOUND YOUR PATTERN! IT STARTS AT CHARACTER 'I3' AND ENDS AT CHARACTER 'I3/
 + 5X'THE TEXT WHICH MATCHES YOUR PATTERN IS:      '7A2//
 + 1X'Once again, 'Z)
  GOTO 2

7 TYPE 'THE PATTERN YOU GAVE ME CANNOT BE FOUND IN THE EXAMPLE TEXT <15>','<15>'
  GOTO 2

8 TYPE 'YOU GAVE ME BAD START/FINISH POSITIONS TO SEARCH THE TEXT <15>','<15>'
  GOTO 2

9 TYPE '<15>','<15>','BYE <15>'
  STOP
  END
```

x x x