IMPLEMENTATION OF SNOBOL4 PATTERN MATCHING

by

JOSEPHINE LI-MING LIU

B.A., Fu Jen Catholic University,

Taipei, Taiwan, ROC 1969

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1977

Approved by:

Linda Shapiro
Major Professor

ACKNOWLEDGEMENT

I would like to thank Dr. Linda G. Shapiro for her advice and guidance throughout the entire project.

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Chapter 1 Introduction

1.1 SNOBOL4 Pattern Matching..... 1

1.2 Goal of the Project..... 7

Chapter 2 Overview of the program

2.1 Major Data Structure..... 10

2.2 Parsing Strategy..... 16

2.3 Pattern Matching Algorithm..... 22

2.4 General Organization..... 26

Chapter 3 Documentation of the Program

3.1 Description of variables..... 28

3.2 Description of Functions

3.2.1 MAIN2..... 33

3.2.2 INITIA..... 33

3.2.3 PARSER..... 33

3.2.4 MATCH..... 37

Chapter 4 Using the Program

4.1 Input Specification..... 41

4.2 Tracing and Debugging Facilities..... 43

4.3 Implementation of the Program..... 47

4.4 Testing of the Program..... 48

Chapter 5 Summary..... 49

REFERENCES..... 51

Appendix A: Program Listing..... 52

Appendix B: Test Data..... 82

LIST OF FIGURES

Number	Title	Page
Figure 1.	Illustration of the PATLINK data structure and the array ARGLIST.....	14
Figure 2.	Step by Step illustration of the status of PATLINK, ARGLIST, ASTK and CSTK.....	18

Chapter 1 Introduction

1.1 SNOBOL4 Pattern Matching

SNOBOL4 is a computer programming language containing many features not commonly found in other programming languages. The basic data element of SNOBOL4 is a character string. The language has operations for joining and separating strings, for testing their contents and for replacing substrings. Using SNOBOL4, strings representing sentences can be broken into phrases or words, and formulas can be broken into components and assembled in another format. Because of its power in string manipulation, SNOBOL4 has become a useful tool in areas such as compilation techniques, machine simulation, symbolic mathematics, text preparation, natural language translation, linguistics, and music analysis [1].

Pattern matching is a fundamental operation in the SNOBOL4 language. It is the process of examining a subject string for the occurrence of a substring specified by a SNOBOL4 pattern. A pattern can be as simple as a character string, for example, 'THIS IS A SNOBOL4 PATTERN.'. During pattern matching, the SNOBOL4 system will try to find the leftmost occurrence of this string in a subject string. A pattern can also be composed of simple SNOBOL4 primitive functions. For example, the primitive pattern 'LEN(4)' matches a character string of length 4.

Two operators, alternation and concatenation, can be used to build pattern structures that match sets of strings. Alternation is indicated by an expression of the form

P1 | P2

where P1 and P2 are simple SNOBOL4 patterns. The value of the expression is a pattern structure that matches any string spceified by either P1 or P2. Concatenation of two pattern, P1 and P2, is specified as

P1 P2.

The value of the expression is a pattern that matches a string consisting of two substrings, the first matched by P1 and the second by P2.

By using alternation and concatenation operators, one can build more complicated pattern structures that match large numbers of strings. Concatenation has higher precedence than alternation, but parentheses can be used to override the built-in precedence of the patterns. Thus, the pattern

'A' | 'B' 'C' | 'D'

matches one of the three strings: 'A', 'BC', and 'D'. However the pattern

```
('A' | 'B')  ('C' | 'D')
```

matches one of the four strings: 'AC', 'AD', 'BC', and 'BD'.

Before matching actually occurs, a pattern is evaluated. Its value is a pattern structure which is used to drive a pattern matching procedure called the scanner which performs the matching. Should any string specified by the pattern appear as a substring of the subject string, pattern matching succeeds. Only the first (leftmost) occurrence of the set of substrings specified matches the pattern. Once pattern matching succeeds, no more alternatives are sought.

The scanner uses two pieces of information during scanning:

1. The cursor:
   A pointer to the position in the subject string where the scanner should be searching for a match to the current pattern component.

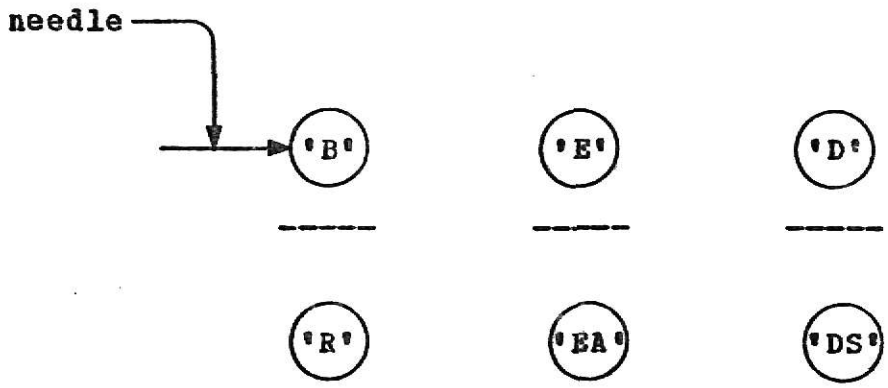
2. The needle:
   A pointer to the current component of the pattern structure.

It is convenient to think of components of a pattern structure as a set of beads that the scanner is trying to thread using a needle. Griswold's [2] illustration of a

bead diagram representing the pattern structure

```
('B' | 'R') ('E' | 'EA') ('D' | 'DS')
```

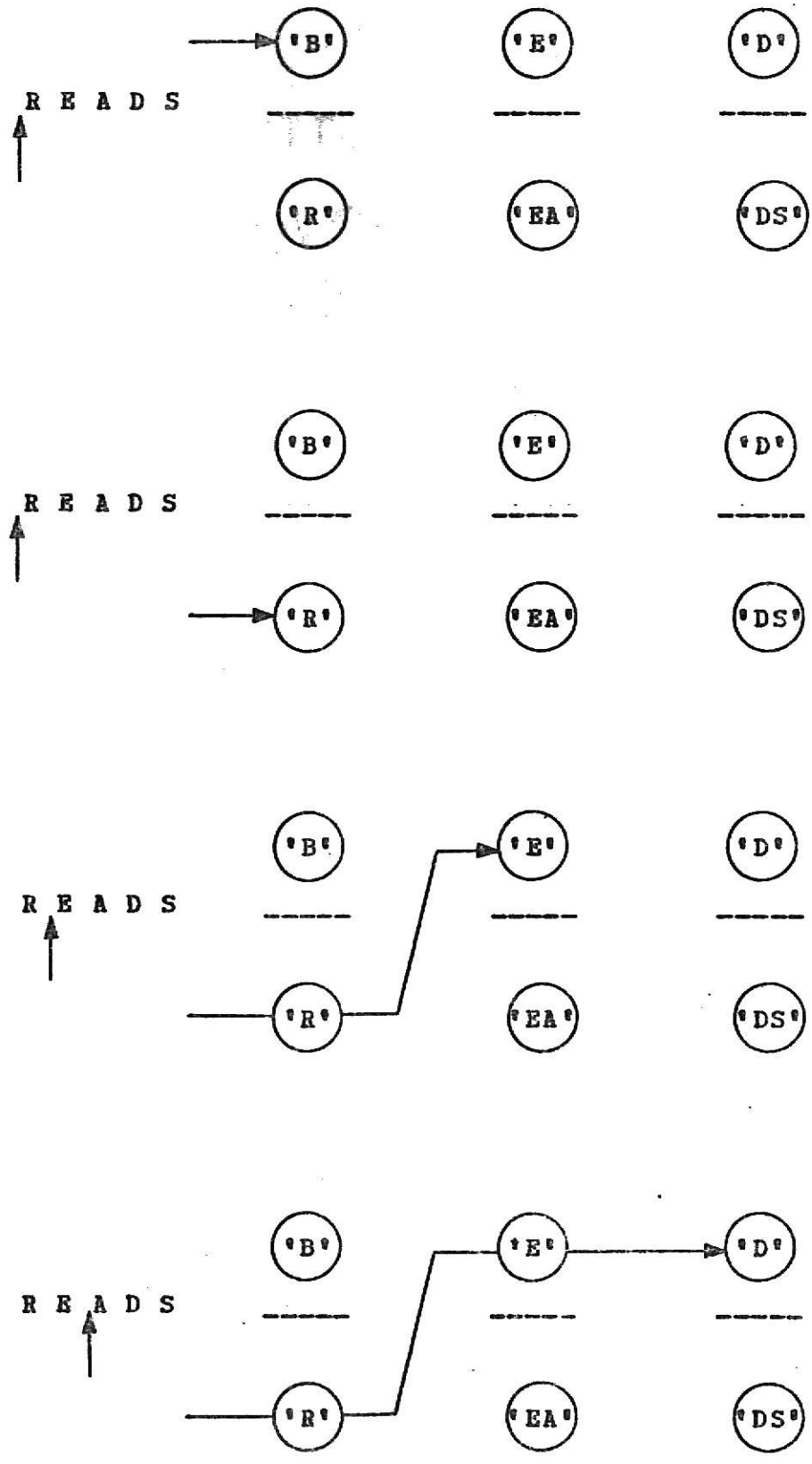
is shown below:

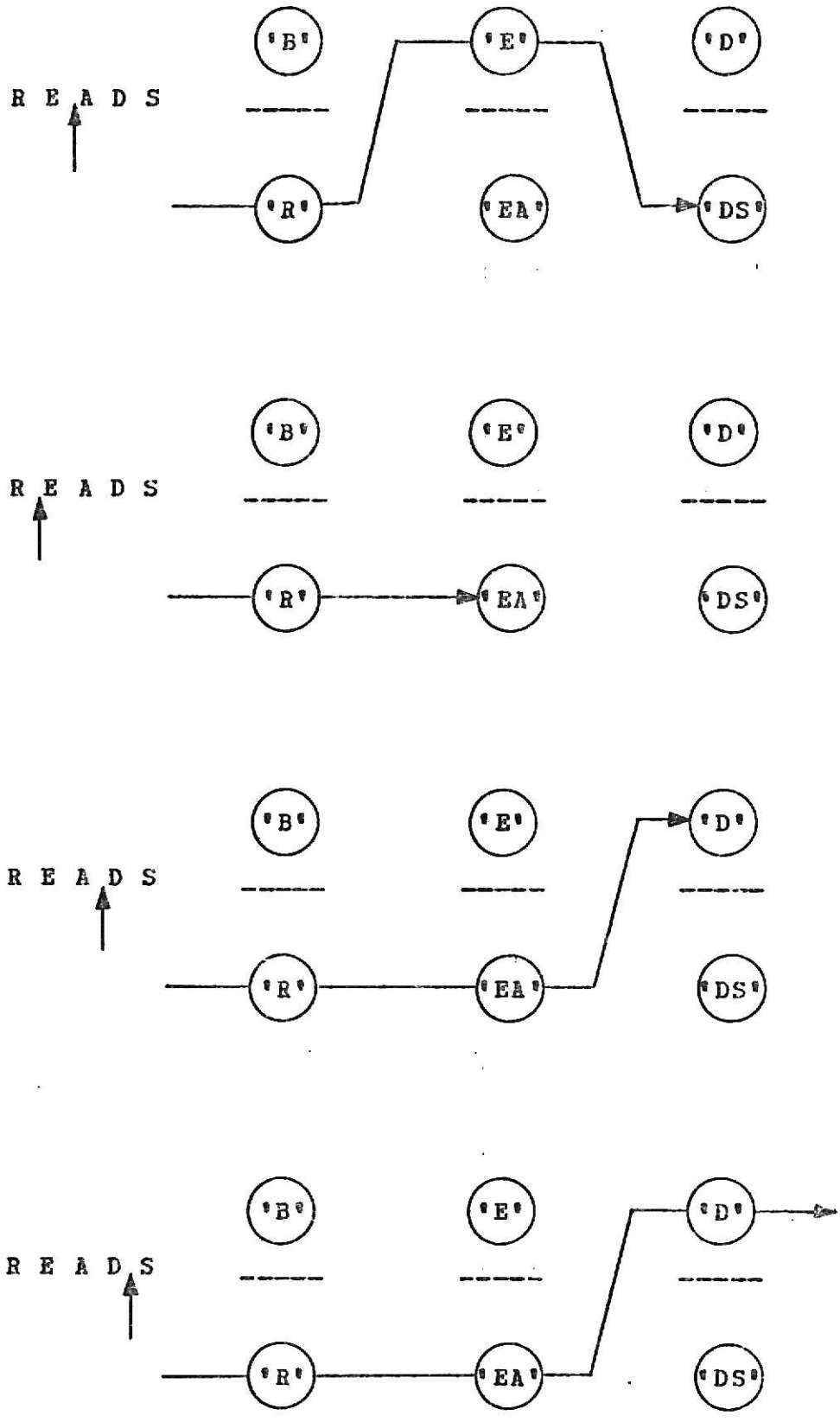


In a bead diagram, left-to-right order of concatenation is preserved. Alternation is represented top to bottom. The needle points at the bead (component) that the scanner is currently trying to match. If a bead matches, the needle passes through and moves upward as far as it can go without crossing a horizontal line. If a bead does not match, the needle moves down to an alternative bead. If no alternative exists, the needle is pulled back through the last successfully matched bead, and an alternative is sought there.

The following figure illustrates the steps in matching the above pattern against the subject string 'READS'. The arrow pointing at 'READS' represents the cursor, while the arrow pointing at the beads represents the needle. Failure

in the fifth step causes the needle to be pulled back. The cursor is moved back at the same time.





In this case, a successful match requires eight steps.

Bead diagrams graphically illustrate that the alternatives are matched in top to bottom order. But in actual SNOBOL4 programs, alternatives are matched by the scanner in left-to-right order. Thus, the scanner attempts to match B before R, E before EA, and D before DS.

1.2 Goal of the Project

The goal of this project is to design and implement a program that performs parsing and pattern matching for a subset of SNOBOL4 patterns. It reads a SNOBOL4 pattern, evaluates it and creates a pattern structure. Then, using the pattern structure, it performs pattern matching on a subject string. When an error occurs, such as illegal pattern detected during parsing, the program will terminate with an error message. Otherwise at the end of each parsing and pattern matching task, the program will print out a message indicating whether or not pattern matching succeeded and, if it succeeded, the substring matched.

The SNOBOL4 pattern features covered in this project include:

1. Character string matching
2. Primitive Functions
   a. LEN(N)
   b. BREAK(STRING)
   c. SPAN(STRING)

3. Alternation and concatenation operators

4. Parentheses

1, 3, and 4 have been discussed earlier in this chapter. The following is a brief description of the three SNOBOL4 primitive functions in 2.

LEN(N) matches any string of the specified length N. The argument N of LEN must have a non-negative integer value when pattern matching is performed. For example, 'LEN(5)' will match a substring of five characters beginning at the current cursor position. Pattern matching will fail if the subject string contains less than five characters.

BREAK(STRING) matches runs of characters which do not appear in the character string STRING. The argument of BREAK must be a nonnull string. Pattern matching succeeds when the first character contained in the argument field is encountered. If the cursor is positioned immediately to the left of a break character, BREAK matches the null string. BREAK must find a break character, or it fails. For example, if the subject string has the value 'THIS IS A SENTENCE.' and the pattern is "BREAK('.')", when pattern matching succeeds, the cursor will be pointing at the left of the period and the substring 'THIS IS A SENTENCE' will match the pattern. Pattern matching will fail if no period appears in the subject string.

SPAN(STRING) matches runs of characters specified in

the character string STRING. The argument field of SPAN must be a nonnull string. For example, the pattern "SPAN('ABCDEFGHIJKLMNOPQRSTUVWXYZ')" will match any substring that contains only alphabetic characters. When a character other than the ones specified in the argument is reached, the scanner will terminate the pattern matching. The cursor will advance to the right of the last character matched by SPAN. SPAN must match at least one character, or it fails.

There are over a hundred SNOBOL4 functions, operators and features. Only a few are covered in this project due to limited time. Additional features can be added to the program with little effort. This will be discussed in Chapter 4.

Chapter 2 Overview of the Program

The parsing and pattern matching program was written in PL/I and tested on the IBM 370 computer. Structured programming techniques were used, making coding, debugging and modification easier, faster and less complicated. In this chapter, the major data structure, the parsing strategy, the pattern matching algorithm, and the organization of the program will be described.

2.1 Major Data Structures

The parsing module of the program (referred to as the parser) scans the characters in the pattern from left to right and recognizes each component and operator. Alternation and concatenation of patterns produce structural relationships between pattern components. These relationships govern the order in which components are matched, depending on the success or failure in matching individual components. The relationships between pattern components are shown in the bead diagram in Chapter 1 in which the alternate-concatenate structure is illustrated. Each component of a pattern corresponds to a primitive pattern-matching operation. Each component is represented by one block of a data structure, PATLINK, which contains the offset (logical address) of the component, and offsets to the alternate and subsequent of the component. A block also contains the name of the function that performs the

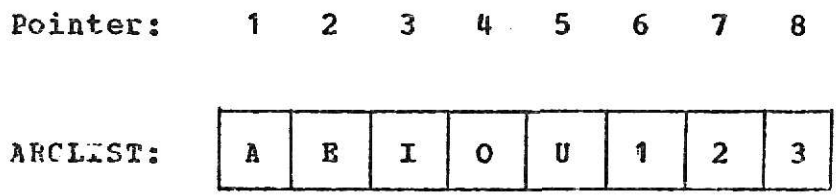
primitive pattern matching operation, the length of the argument, and a pointer to the argument. The information in PATLINK is used during pattern matching. The design of the PATLINK data structure is based on the structures described in Griswold [3] and Gimpel [4]. PATLINK and the other major data structures used in the program will be discussed in detail in the remainder of this section.

ARGLIST

ARGLIST is an array of 80 elements which stores the characters from string patterns and argument lists of primitive functions. Because the argument of each pattern component is of a different length, it is difficult to store them in the PATLINK data structure. Instead, pointers are included in the data structure pointing to the positions in ARGLIST where the argument of each component is stored. For example, the pattern

```
SPAN('AEIOU') | '123'
```

will cause the following ARGLIST to be generated.



The pointer to the ARGLIST in the first component will be

logical address 1 and the length will be 5. The pointer in the second component will be logical address 6 and the length will be 3.

PATLINK

PATLINK is declared in the main routine, MAIN2, as a controlled variable. Its size can shrink and grow accordingly. As described above, each element of PATLINK represents a component in the pattern and has the following structure:

NUM	PAT
CAT	ALT
ARG	LEN

Where

NUM - contains the logical address of the pattern component in the linked data structure.

PAT - is the function name of the component, i.e. either LEN, BREAK, SPAN or STRING.

CAT - contains the logical address of the subsequent component to be matched after the present pattern component succeeds in matching a substring of the subject string. If there is no subsequent component,

the value of this field is 0.

ALT - holds the logical address of the first component of the next alternate pattern. If there are no more alternates, the field is 0.

ARG - is a pointer to the starting position of the arguments of the pattern which are stored in ARGLIST.

LEN - contains the length of the string argument of the pattern stored in ARGLIST.

Each individual primitive component in the pattern structure from left to right is assigned a value from one to the number of primitive components in the pattern as its logical address in the data structure. For example, in the pattern

```
('AB' | LEN(4))  BREAK('.'),
```

there are three components. 'AB' is assigned the logical address 1, LEN(4) is assigned the logical address 2 and BREAK('.') is assigned the logical address 3.

Figure 1 is an illustration of the PATLINK data structure and the array ARGLIST after the parsing of the above pattern. LEN does not use any space in ARGLIST because the argument of LEN is an integer which is stored

directly in PATLINK. The parser puts the value 4, which is the specified length, in the ARG field of the component LEN(4). The LEN field is 0.

Pattern - ('AB' | LEN(4)) BREAK('.')

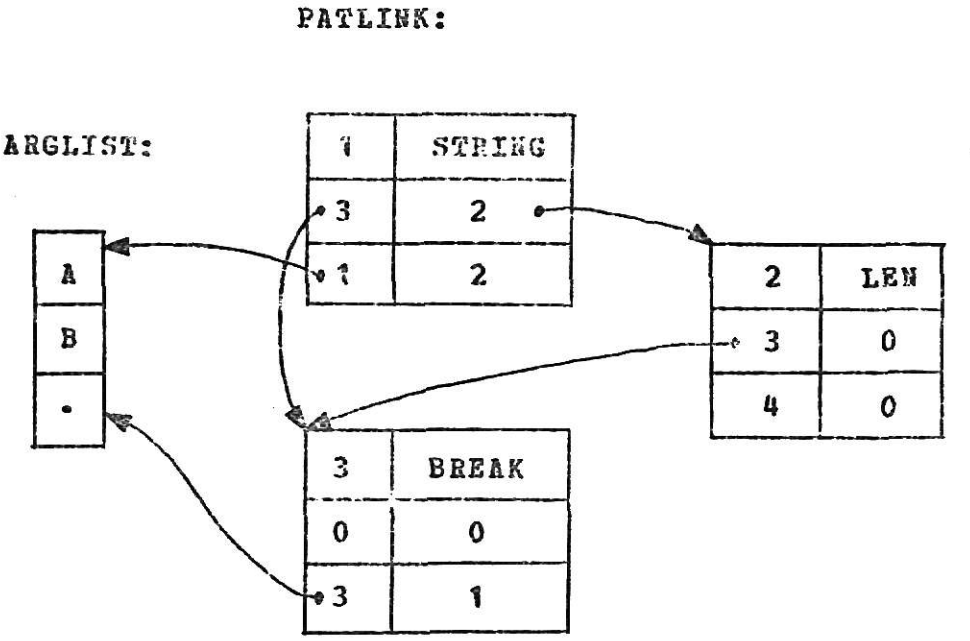


Figure 1.

PATTAB

PATTAB is a table declared and initialized in the main routine, MAIN2. It holds the names of the primitive functions and the labels in the program of statements which call the pattern matching subroutines corresponding to these functions. The size of PATTAB is the number of primitive functions to be implemented. PATTAB is used by the parser to perform a table look-up whenever a primitive function is encountered. If the primitive function is not in the table, it is an illegal function. The program will print out an error message. Each element of PATTAB has the structure:

where

PRIM - is the name of the primitive function.

ARG - has the value '1'B if the primitive function has an argument, otherwise this field is '0'B.

LBV1 - contains a label which will be used later by subroutine MATCH to determine which subroutine should be called to perform pattern matching.

In the current implementation, the size of PATTAB is 3, and it contains the following information:

LEN	1	FLEN
SPAN	1	FSPAN
BREAK	1	FBREAK

FLEN, FSPAN and FBREAK are labels in the program.

2.2 Parsing Strategy

There are two stacks used in subroutine PARSER. Their use was motivated by Gimpel [4].

CSTK - contains the logical address of pattern components waiting for a concatenation and left and right parentheses.

ASTK - contains the logical address of pattern components waiting for an alternate and left and right parentheses.

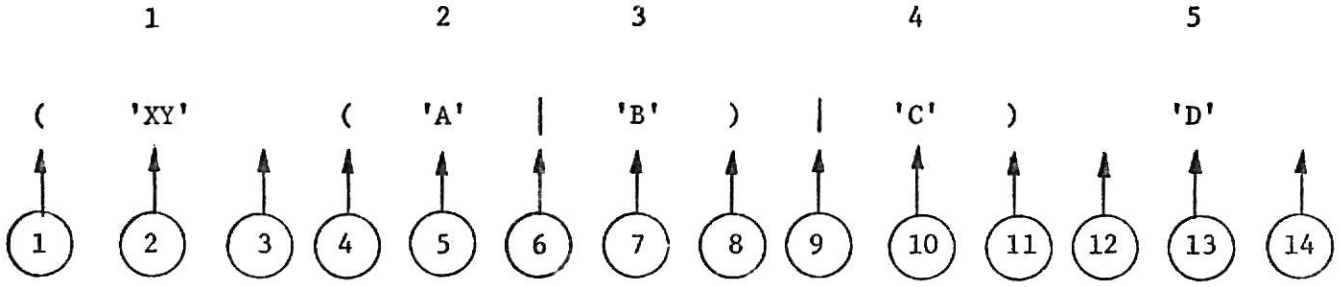
During the process of parsing, when a left parenthesis is reached, the parser will first push the left parenthesis onto both ASTK and CSTK, and then push the logical address of the pattern component immediately following the left parenthesis onto both ASTK and CSTK. When a right parenthesis is encountered, the parser will check the operator immediately following the right parenthesis. If it

is an alternation operator, the parser will pop everything up to and including the left parenthesis in ASTK and link the next component to the one on top of ASTK if ASTK is not empty. If the operator immediately following the right parenthesis is a concatenation operator, the parser will pop everything up to and including the left parenthesis in ASTK and link the next pattern component with all the components in CSTK up to the left parenthesis before popping CSTK. The exact algorithm will be given in Chapter 3.

Figure 2 shows a step by step illustration of the status of PATLINK, ARGLIST, ASTK and CSTK while parsing the pattern

('XY' ('A' | 'B') | 'C') 'D'.

The numbers above the pattern are the logical addresses of the components to be pushed on the stacks. The numbers in circles with arrows under the pattern represent each step of parsing.



Step 1

	ASTK	CSTK	PATLINK	ARGLIST
1	(	(	Empty	Empty

Step 2

	ASTK	CSTK	PATLINK		ARGLIST	
2	1	2	1	STRING	1	X
1	(	1	-	-	2	Y
			1	2		

Step 3

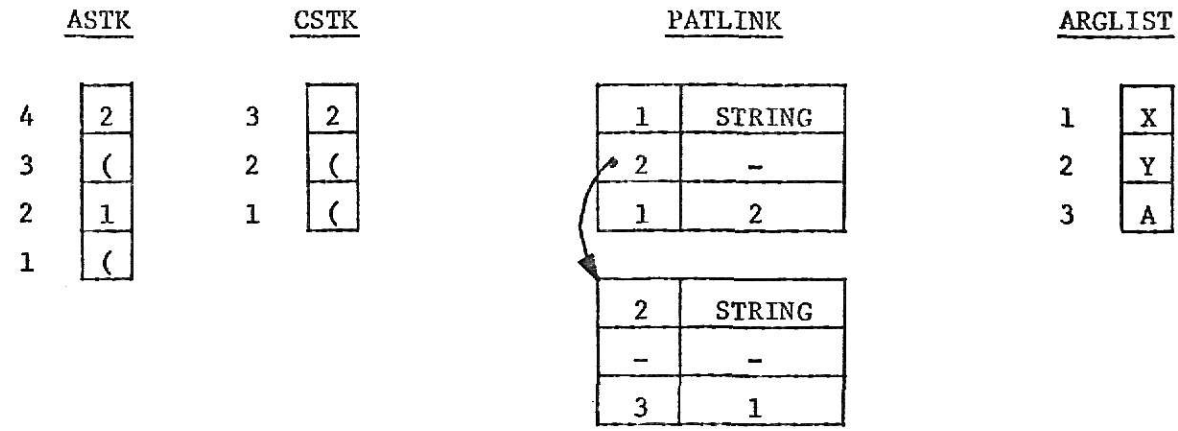
	ASTK	CSTK	PATLINK		ARGLIST	
2	1	1	1	STRING	1	X
1	(	(	2	-	2	Y
			1	2		

Step 4

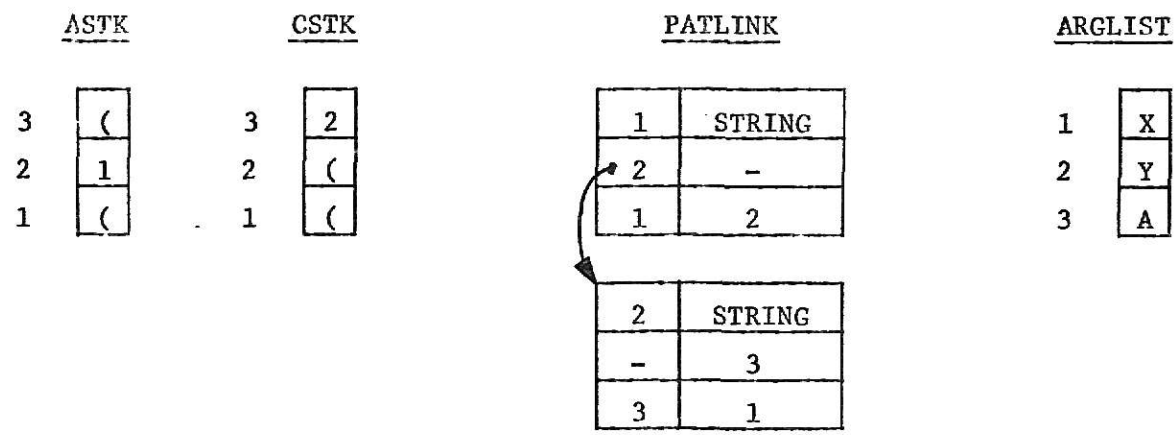
	ASTK	CSTK	PATLINK		ARGLIST	
3	(	2	1	STRING	1	X
2	1	1	2	-	2	Y
1	(	(	1	2		

Figure 2.

Step 5



Step 6



Step 7

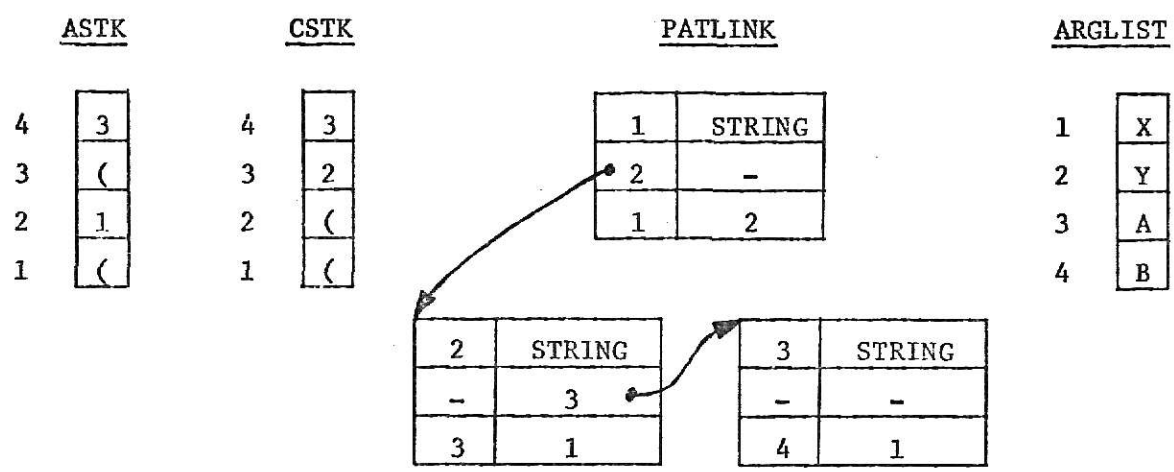
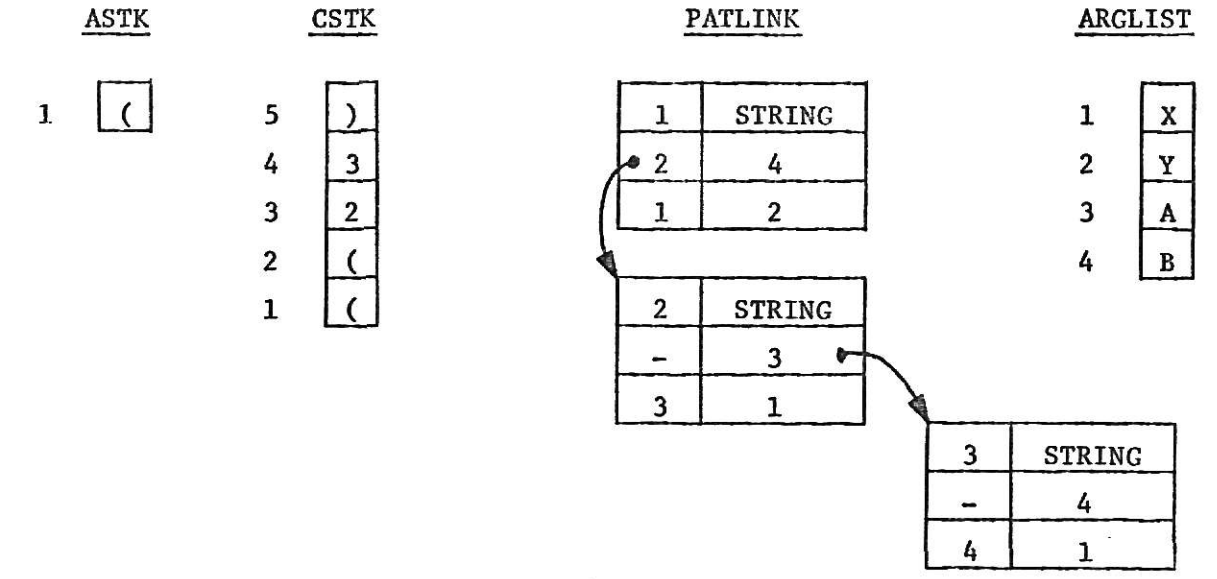


Figure 2. (Continued)

Steps 8 and 9



Step 10

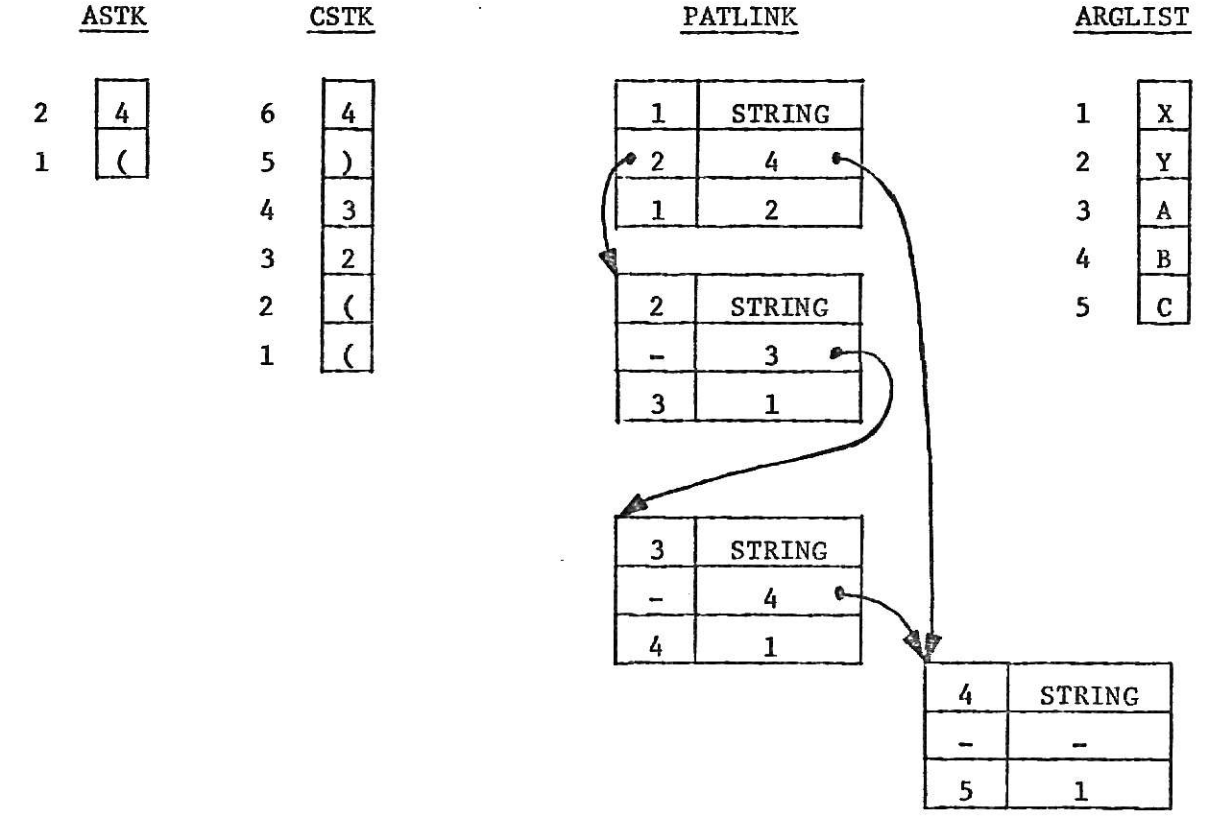
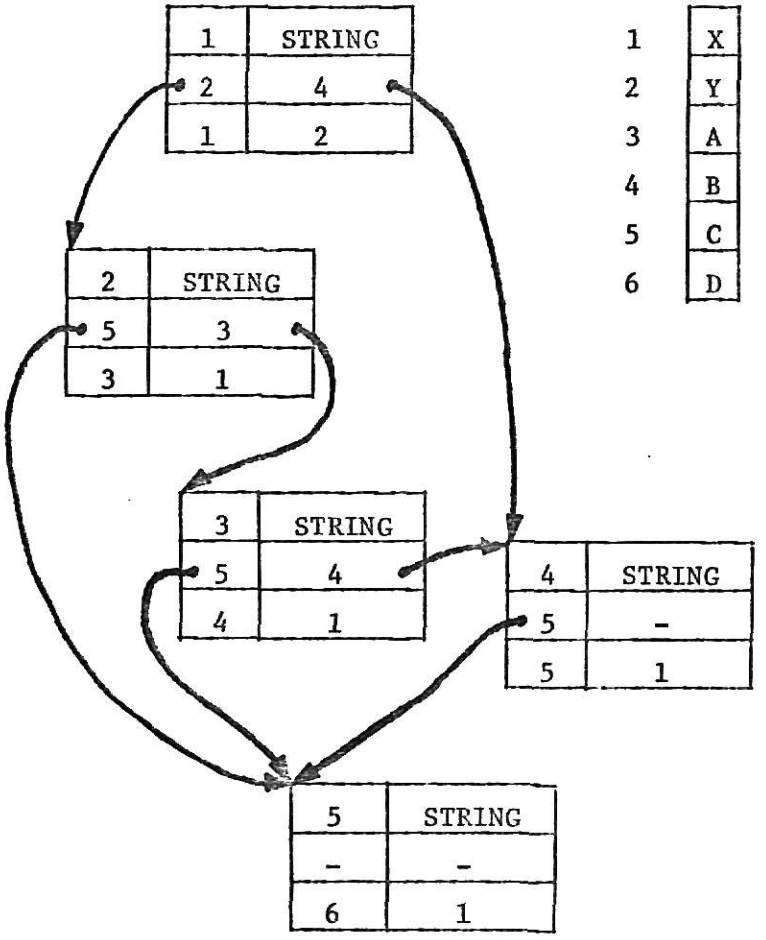


Figure 2. (Continued)

Step 11 and 12

ASTK	CSTK	PATLINK	ARGLIST
Empty	Empty		

Step 13

	ASTK		CSTK	PATLINK	ARGLIST
1	5	1	5	Same	Same

Step 14

ASTK	CSTK	PATLINK	ARGLIST
Empty	Empty	Same	Same

Figure 2. (Continued)

2.3 Pattern Matching Algorithm

Subroutine MATCH does the actual pattern matching for the program. MATCH uses two pointers, NEEDLE and CURSOR, while performing pattern matching. Both NEEDLE and CURSOR work similarly to the needle and the cursor in the scanner discussed earlier in Chapter 1.

When pattern matching starts, NEEDLE points to the first component in PATLINK and CURSOR points to the beginning of the subject string. If matching succeeds, NEEDLE will move to the subsequent component, and CURSOR will advance the number of characters the current pattern matching successfully matched. When matching fails, an alternate component will be sought. However, if the position in the subject string pointed to by CURSOR matches none of the first components of all the alternate patterns, CURSOR will be advanced by one position and pattern matching will start over from there.

Subroutine MATCH uses two stacks to save the status of NEEDLE and CURSOR.

CPOS - holds the number of positions CURSOR has been advanced in each successful match.

ASTK - This ASTK is different from the one in PARSER. It contains ALT fields of the current successfully matched components in case NEEDLE has to be backed up

when a subsequent match fails. A detailed pattern matching algorithm will be given in Chapter 3.

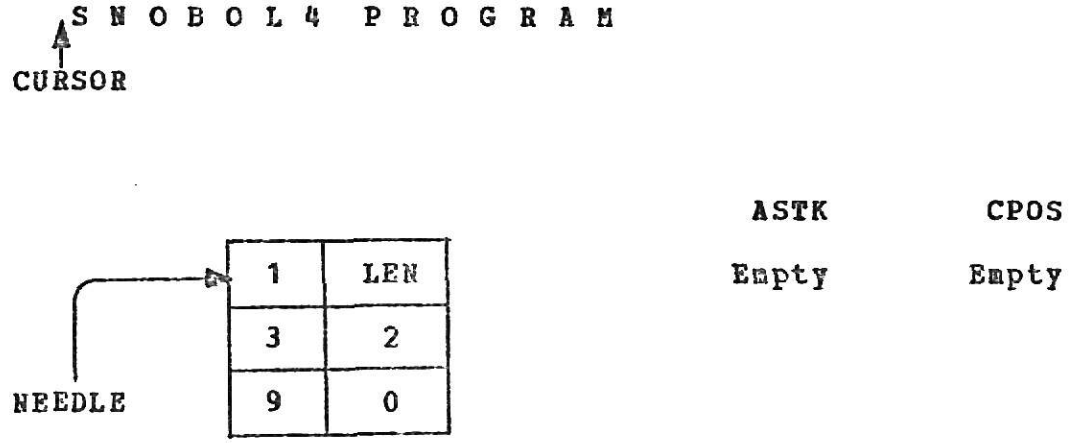
The following is an illustration of each step of the pattern matching operation, including the positions of NEEDLE and CURSOR and the status of CPOS and ASTK. In this example the subject string

'SNOBOL4 PROGRAM'

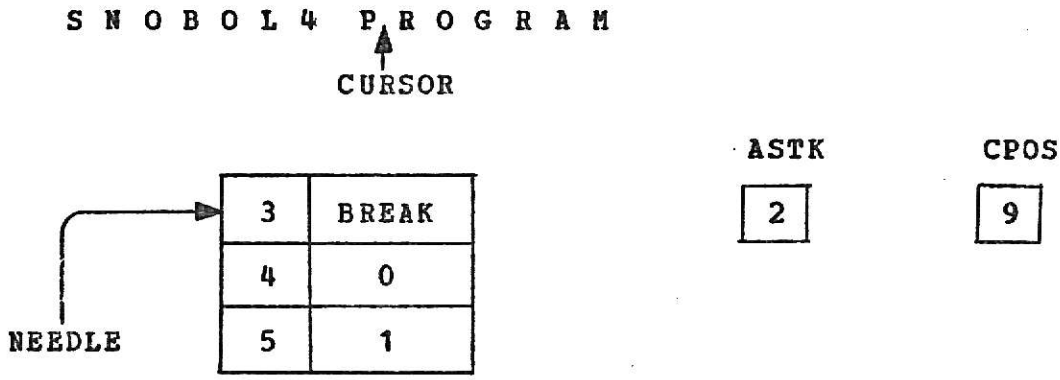
is searched for an occurrence of the pattern

(LEN(9) | 'SNOB') BREAK('4') SPAN('4').

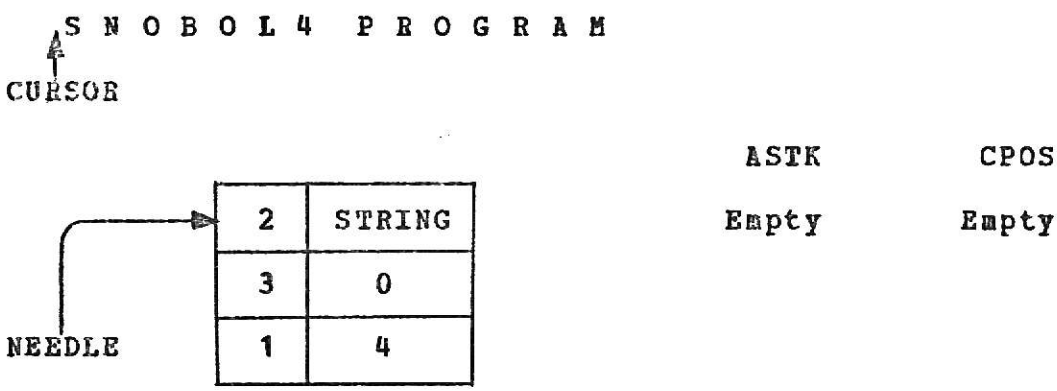
Step 0 - NEEDLE points to the first component in PATLINK, CURSOR points to the beginning of the subject string.



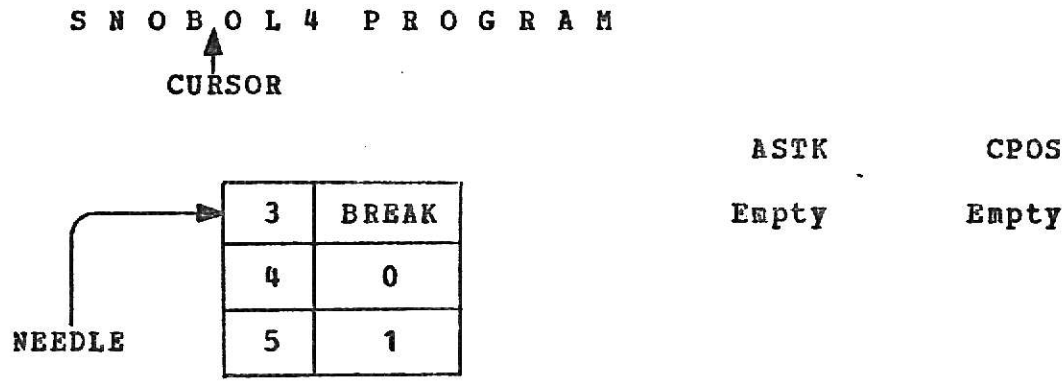
Step 1 - LEN(9) matches the substring 'SNOBOL4 P'. A subsequent component is sought. The logical address of the alternate component(2) is pushed on ASTK. The value 9 is pushed on CPOS. CURSOR is moved to point to the tenth character in the subject string.



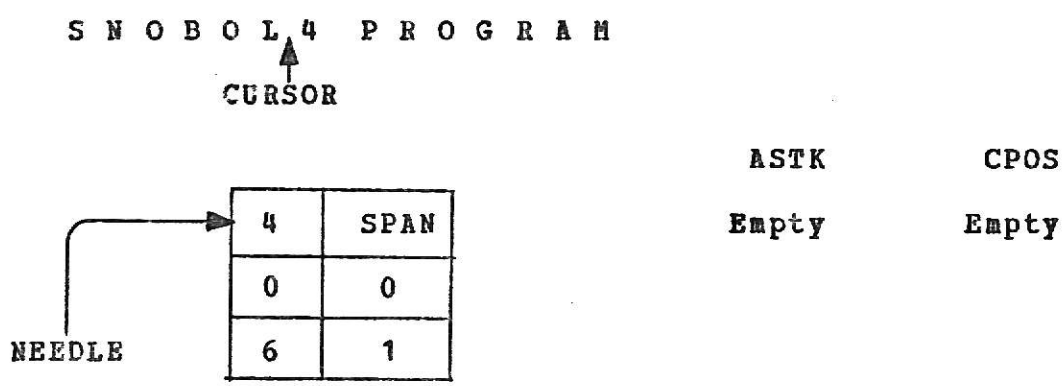
Step 2 - The pattern matching routine tries to find a break character '4' but fails. NEEDLE has to be pulled back and assigned the value on top of ASTK, and CURSOR is also backed up 9 spaces.



Step 3 - Pattern matching matches the substring 'SNOB'. The ALT field in the current PATLINK element is 0, so no action is taken with respect to the stacks CPOS and ASTK. A subsequent component is sought.



Step 4 - Pattern matching successfully locates the break character. BREAK('4') matches the substring 'OL'. A subsequent is sought.



Step 5 - SPAN matches the character '4'. There are no more subsequent components which indicates that pattern matching has succeeded.

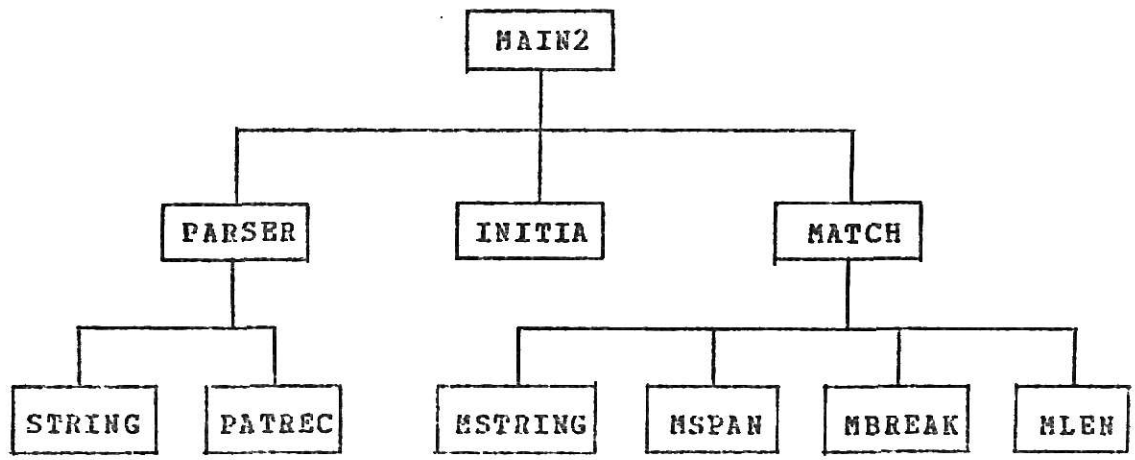
S N O B O L 4 P R O G R A M
CURSOR

Step 6 - The substring 'SNOBOL4' matches the pattern.

2.4 General Organization

The program contains a main routine which serves as a driver and nine subroutines. The major tasks of the program include a parser and a pattern matching subroutine.

The following diagram shows the hierarchical structure of the program.



All the subroutines are internal to MAIN2. PARSER and MATCH are unknown to each other. All the variables are used according to their scope, therefore, each subroutine is constructed without a parameter list and invoked without an argument list. A detailed description of each subroutine is given in Chapter 3.

Chapter 3 Documentation of the Program

This implementation of the program contains two major routines, PARSER and MATCH. PARSER serves as a lexical analyser and constructs the pattern structure. MATCH is the pattern matching subroutine. Both PARSER and MATCH are invoked by a driver MAIN2. There is one small subroutine INITIA called by MAIN2 to initialize the variables. The general flow of the program starts from MAIN2. MAIN2 invokes INITIA to initialize the global variables and then invokes PARSER to recognize the functions and operators and create the data structure, PATLINK. After PARSER returns to MAIN2 with no error, MAIN2 will call MATCH to perform pattern matching on a subject string. If an error occurs in either PARSER or MATCH, the return code RCODE is set to a value other than 0. The value of RCODE depends upon the different error conditions and is checked upon return to MAIN2. The program prints an error message for non-zero RCODE and goes on to the next pattern matching task.

3.1 Description of Variables

All the global variables are declared in MAIN2. Their names, attributes and uses are described in this section.

PATTERN is declared as a controlled variable array of N elements. Each element has the attribute CHARACTER(1). PATTERN is used to hold a SNOBOL4 pattern. The value of N

depends upon the size of the pattern. Because the pattern can be very short and it can also be very long. In order to save storage, PATTERN is declared as a controlled variable to be allocated at the time of execution.

PATLINK was discussed in detail in Chapter 2. The attributes of each field of PATLINK are:

NUM - FIXED(2)

PAT - CHARACTER(6)

CAT - FIXED(2)

ALT - FIXED(2)

ARG - FIXED(2)

LEN - FIXED(2)

LBV is a controlled variable array having the same dimension as PATLINK after allocation. Each element of LBV has the attribute CHARACTER(7) and contains the label of a statement in the program that the corresponding component in PATLINK will branch to in order to initiate the correct function call. For example, if PATLINK(2).PAT has the value 'STRING', LBV(2) will have the value 'FSTRING' which is the label of the function call in the program.

FLAG1 and FLAG2 are flag bits in the program both having attribute BIT(1). Their values determine whether the program trace will be turned on or off. They will be discussed in detail in Chapter 4.

ARGLIST has been covered in detail in Chapter 2. Each

element in ARGLIST has attribute CHARACTER(1). ARGPTR is the pointer to ARGLIST. It has attribute FIXED(2) and is initialized to 0.

PATLINKCTR is the pointer into PATLINK and has attribute FIXED(2). It is initialized to 1 in subroutine INITIA and used by PARSER to create PATLINK.

RCODE is the return code, with initial value 0 for normal continuation of the program. Any value other than 0 will cause an error message to be printed. Following is a list of the error messages for each different error condition returned from PARSER.

RCODE=1 - 'Illegal primitive function'

RCODE=2 - 'No blank between function and operator'

RCODE=3 - 'Unmatched parentheses pair'

RCODE=4 - 'Illegal operator'

RCODE=5 - 'No argument list'

When RCODE=1 is returned from subroutine MATCH, the message 'PATTERN MATCHING FAILED' will be printed.

INSTRING holds the value of the subject string to be matched against the pattern. Its attribute is CHARACTER(50) VARYING.

FIRST is a pointer to the first character in the substring of the subject string matched successfully by the

pattern matching operation. LAST is a pointer to the last character in the substring of the subject string matched. The values of FIRST and LAST are used to locate the substring matched.

EOF is a flag bit with the initial value '0'B. If an end-of-file condition occurs, its value will be set to '1'B and the program will be terminated.

The following are variables local to subroutine PARSER.

CSTK and ASTK were discussed in Chapter 2. They are both of size 4 greater than the size of PATLINK to make sure there is enough room for all the components and parentheses in the pattern. They both have attribute FIXED(2). Because ASTK and CSTK only hold fixed decimal numbers, left and right parentheses which are pushed on ASTK and CSTK during the process of parsing, are assigned the values 98 and 99, respectively.

CURTOK is a variable used by PARSER to temporarily store the name of a primitive function. Its attribute is CHARACTER(8). PTR is a pointer to the array PATTERN and is used by PARSER to scan the pattern. Its attribute is FIXED(3).

A98 and A99 are counters of left parenthesis and right parenthesis, respectively, in ASTK. They both have

attribute FIXED(2). If the value of A98 is not equal to A99 at the end of parsing, it indicates an unmatched parentheses pair, and RCODE will be set to 3. C98 and C99 are counters of left and right parentheses, respectively, in CSTK. They both have attribute FIXED(2). If they have unequal value at the end of parsing, RCODE will be set to 3 for an unmatched parentheses pair. A and C are pointers to ASTK and CSTK, respectively. They both have attribute FIXED(2).

Subroutine MATCH has the following local variables:

CURSOR and NEEDLE have the attribute FIXED(2). Their functions have been covered in Chapter 2. CPOS and ASTK have been discussed in Chapter 2. Both have attribute FIXED(2), and each stack is of size 10. APTR and CUPTR are pointers to the stacks ASTK and CPOS, respectively. Both have the attribute FIXED(2).

BINGO has the attribute BIT(1) and is used by MATCH as a flag bit. Its value is initialized to '0'B. When the first matching operation succeeds, it is set to '1'B, and the current value of CURSOR is assigned to FIRST to mark the beginning of the substring matched.

3.2 Description of Functions

3.2.1 MAIN2

MAIN2 is a main routine which declares all global variables, does most of the output and the input, allocates all the controlled variables, and checks RCODE. It mainly serves as a driver for two major subroutines, PARSER and MATCH.

3.2.2 INITIA

INITIA is a subroutine called by MAIN2 to initialize the global variables. These variables are CAT, ALT and LEN in PATLINK, FLAG2, ARGPTR, PATLINKCTR and RCODE.

3.2.3 PARSER

There are two subroutines called by PARSER:

1. STRING is invoked whenever a string component is encountered. It adds the characters between a pair of quotes to ARGLIST, fills in the data structure PATLINK with the proper information, and increments the counter variables PTR and ARGPTR, before returning to the calling procedure.

2. PATREC is invoked when a possible primitive function is detected. It takes the name stored in CURTOK, and performs a table look-up in PATTAB to make certain the primitive

function is legal. Then PATREC fills in PATLINK and increments counter variables PTR and ARGPTR, before returning to the calling routine. If an error condition occurs, such as an illegal function or no argument lists, the return code RCODE will be set before return and no other actions will be taken.

PARSER uses the index variable PTR to scan through the array PATTERN and to recognize each pattern component and operator. There are five cases corresponding to the five syntactic items that PTR can point to. These five cases are included in a DO WHILE loop which tests the pattern pointer PTR against the length of the pattern PATTERN. The first four cases are implemented in nested IF-THEN-ELSE clauses, so when one case is false the ELSE path will be taken. The fifth case is the case for operators, and will be executed after one of the preceeding four cases has been completed. The five cases are:

CASE 1 - a left parenthesis.

PARSER will push the value 98 on both ASTK and CSTK and skip any blanks that may follow until a non-blank element is reached. Only three possibilities are allowed after the left parenthesis:

1. Another left parenthesis - no action.
2. A string - PARSER will call subroutine STRING.
3. A primitive function - PARSER will call subroutine

PATREC.

For possibilities 2 and 3, PARSER will push the current value of PATLINKCTR (the logical address of the current component) on both ASTK and CSTK and increment PATLINKCTR by 1.

CASE 2 - a quote.

PARSER will call subroutine STRING. Upon returning from STRING, PARSER will push the current value of PATLINKCTR on both ASTK and CSTK and increment PATLINKCTR by 1.

CASE 3 - a right parenthesis.

PARSER will look ahead for any one of the following four possibilities and perform the subsequent actions:

1. End of PATTERN - PARSER will push the right parenthesis (99) on both ASTK and CSTK.

2. Another right parenthesis - PARSER will pop everything up to and including a left parenthesis from ASTK, then push the right parenthesis on CSTK.

3. An alternation operator - PARSER will push the right parenthesis on CSTK, then pop everything out of ASTK up to and including a left parenthesis. If ASTK

is not empty, PARSER will put the current value of PATLINKCTR in the ALT field of the pattern component on top of ASTK and pop the top of ASTK.

4. A concatenation operator - PARSER will pop everything up to and including a left parenthesis from ASTK and put the current value of PATLINKCTR in the CAT field of all the components in CSTK up to a left parenthesis.

CASE 4 - a primitive function.

If none of the above three cases applies, the case has been narrowed down to a primitive function. PARSER will call PATREC to recognize the primitive function and fill in PATLINK.

CASE 5 - an operator, either concatenation or alternation.

If it is a concatenation operator, PARSER will pop a component from the top of CSTK and link the CAT field of that component with the next component. If it is an alternation operator, PARSER will pop the first element from ASTK and links its ALT field with the next component.

While popping CSTK and ASTK in 3 and 4 of CASE 3, PARSER makes sure the right and left parentheses are balanced by using the counter variables A98, A99, C98 and

C99. Whenever a left parenthesis in CSTK or ASTK is popped, the value of C98 or A98, respectively, will be incremented by 1. Whenever a right parenthesis in CSTK or ASTK is popped, the value of C99 or A99, respectively, will be incremented by 1. If A98 and C98 are not equal to A99 and C99, respectively, at the end of parsing the pattern, RCODE will be set to 3.

In all the above cases, PARSER checks for at least one blank between operators and pattern components. If there is no blank, RCODE is set to 2.

3.2.4. MATCH

Subroutine MATCH performs the pattern matching for the program. There are four subroutines called by MATCH.

1. MSTRING does the actual pattern matching for a string pattern. It uses the value in the ARG field of the PATLINK element that NEEDLE is currently pointing to as the beginning pointer to ARGLIST. It then extracts the number of characters specified in the LEN field from ARGLIST. Finally, it compares the string from ARGLIST with the substring of the subject string pointed to by CURSOR. If the substring successfully matches the string pattern, CURSOR will be advanced to the right of the last character in the substring matched.

2. MSPAN performs the actual pattern matching for the primitive function SPAN. It checks each character in the substring of the subject string pointed to by CURSOR against the argument string stored in ARGLIST. If the current character appears in the argument string, CURSOR is advanced to the next character in the subject string. The process continues until a character which does not appear in the argument string is reached or the end of the subject string is reached. Pattern matching succeeds if at least one character matches. The CURSOR is advanced to the left of the character which is not in the argument string.

3. MBREAK performs actual pattern matching for the primitive function BREAK. It starts at the character in the substring of the subject string pointed to by CURSOR. If the character is not in the argument string, CURSOR will be advanced to the next character. The process continues until a character in the subject string is found in the argument string. Pattern matching succeeds when a break character is found. The CURSOR is positioned at the left of the break character when pattern matching stops.

4. MLEN performs actual pattern matching for the primitive function LEN(N). It checks the length of the subject string from the character pointed to by the CURSOR. If this substring has N or more characters, pattern matching succeeds, and CURSOR is moved N positions to the right of its current position.

For the above four subroutines, the number of positions CURSOR is advanced in a successful match is saved in the stack CPOS, if the ALT field of the current component is non-zero. If pattern matching fails, CURSOR will stay at the same position as when the current pattern matching operation started, and RCODE will be set to 1.

Before pattern matching starts, a subject string is read into INSTRING. When pattern matching starts, NEEDLE will be pointing to the first component in the PATLINK data structure, and CURSOR will be pointing to the beginning of INSTRING. MATCH searches LBV for the pattern component in PATLINK and decides which subroutine it should call. After return from the subroutine call, MATCH will check the value of RCODE. If matching succeeded (RCODE=0), it will check the value of BINGO. If BINGO is '0'B which means this is the first successful match, the value of CURSOR before the match will be assigned to FIRST to mark the beginning of the substring matched. Then MATCH will check the CAT field of the current component. If it is non-zero, its value will be assigned to NEEDLE for the next matching operation and the non-zero ALT field will be saved on ASTK in case NEEDLE needs to be backed up. A zero value in the CAT field means there are no more subsequent components in the pattern. The value of CURSOR is saved in LAST and control returns to the calling point in MAIN2. If matching fails, MATCH will check the ALT field of the current component for an alternate. If the ALT field is non-zero, NEEDLE will get the value in the ALT field, and RCODE will be reset to 0 before continuing

pattern matching. If the ALT field of the current component is zero, NEEDLE and CURSOR have to be backed up by popping the top of ASTK for an alternate and the top of CPOS for a former CURSOR, provided both stacks are not empty. If there was no previously successful match (BINGO='0'B), NEEDLE will start again from the first component in the pattern, and CURSOR will be advanced by 1 to try again.

MATCH will not be called if the value of RCODE returned from PARSER is non-zero. MAIN2 will detect the non-zero RCODE and print out an error message according to the error condition. If the value of RCODE from MATCH is zero, MAIN2 will print out the substring matched and the message 'PATTERN MATCHING SUCCEEDED.'. If the value of RCODE returned from MATCH is 1, MAIN2 will print out the message 'PATTENRN MATCHNING FAILED.'.

Chapter 4 Using the Program

4.1 Input Specification

This program is designed to perform parsing and pattern matching on sets of test data. The execution of the program will continue until all input data is exhausted. Each set of test data has at least six input values. The following is a description of each input value:

1. The first input is the flag bit FLAG2 which controls the tracing facility. It is punched in the first column of the first data card. The value '0'B turns off the tracing, and the value '1'B turns on the tracing.

2. The second input is the value of N which is the number of characters in the pattern and is used to allocate the controlled variable PATTERN. It should occupy the first three columns of the second input card. The value of N must be the same as the length of PATTERN in order to allocate the correct size array, otherwise, an error will occur later in parsing.

3. The third input is the value of PATTERN. It should start on the second card, directly after the value of N and continue for as many cards as necessary. It will be read in one character at a time and stored in the array PATTERN.

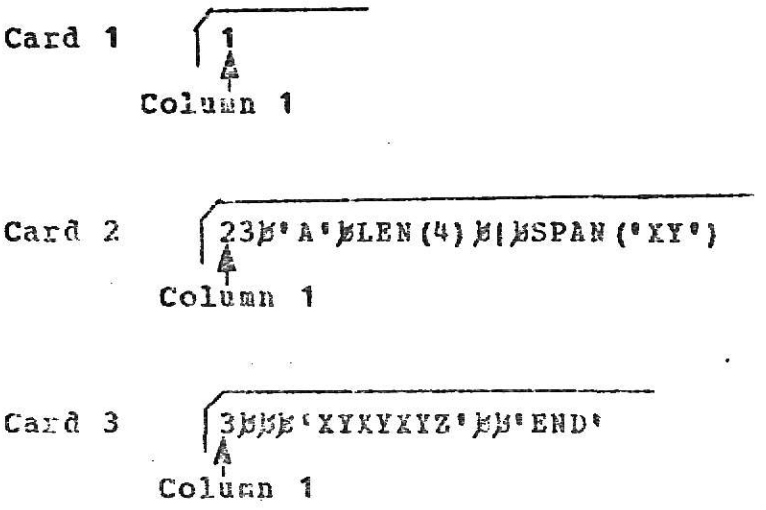
4. The fourth input is the value of M which is the number of components in the pattern. M will be used to allocate the size of PATLINK and LBV. It should appear in the first two columns of the next data card.

5. The fifth input is the value of the subject string, INSTRING. This value should begin on the same data card as M and should be enclosed in a pair of quotes. There can be several input subject strings to match the same pattern structure. The pattern matching procedure will continue for different input strings until a string 'END' is reached.

For example, if the pattern has the value

```
'A' LEN(4) | SPAN('XY')
```

and the subject string is 'XYXYXYZ', and the tracing facility is turned on, the data cards would be



4.2 Tracing and Debugging Facilities

FLAG1 and FLAG2 are used in the program as two one-bit flags. The value of FLAG2 is input from the first data card. FLAG2 controls tracing of the execution of the program. If FLAG2='1'B the program will print out the name of each case being executed, the name of each subroutine called, the status of all stacks CSTK, ASTK and CPOS, and the important variables CURSOR and NEEDLE, at each stage of execution. The value of FLAG1 is dependent on the value of RCODE. If PARSER has been executed without error, upon returning to MAIN2, FLAG1 will be set to '1'B to print out the contents of the PATLINK data structure. FLAT1 will be set back to '0'B before MATCH is called. If MATCH has been executed successfully, FLAG1 will be set to '1'B again to print out the substring matched and the message 'PATTERN MATCHING SUCCEEDED.'.

For example, using the same pattern

```
'A' LEN(4) | SPAN('XY')
```

and the subject string 'XYXYXYZ', if FLAG2='0'B, the program will print out the following:

THE PATTERN TO BE USED IS:

```
'A' LEN(4) | SPAN('XY')
```

1	STRING
FSTRING	
2	3
1	1
2	LEN
FLEN	
0	0
4	1
3	SPAN
FSPAN	
0	0
2	1

THE STRING TO BE MATCHED IS: 'XYXYXYZ'.

SUBROUTINE MATCH MATCHED SUBSTRING 'XYXYXY'.

PATTERN MATCHING SUCCEEDED.

However, with FLAG2 turned on ('1'B), the program will print out:

THE PATTERN TO BE USED IS:

```
'A' LEN(4) | SPAN('XY')


SUBROUTINE PARSER  ENTERING CASE 2 -  SUBROUTINE STRING
CALLED.

SUBROUTINE  PARSER  ENTERING  CASE  5  -  OPERATOR
ENCOUNTERED.

STATUS OF ASTK:
     1

STATUS OF CSTK:

STATUS OF ARGLIST:
     A

SUBROUTINE PARSER  ENTERING CASE 4 -  SUBROUTINE PATREC
CALLED.

SUBROUTINE  PARSER  ENTERING  CASE  5  -  OPERATOR
ENCOUNTERED.

STATUS OF ASTK:

STATUS OF CSTK:
     2

STATUS OF ARGLIST:
     A

SUBROUTINE PARSER  ENTERING CASE 4 -  SUBROUTINE PATREC
CALLED.

STATUS OF ASTK:
     3

STATUS OF CSTK:
     2
     3

STATUS OF ARGLIST:
     A
     X
     Y
```

1	STRING
FSTRING	
2	3
1	1
2	LEN
FLEN	
0	0
4	1
3	SPAN
FSPAN	
0	0
2	2

THE STRING TO BE MATCHED IS: 'XYXYXYZ'.

THE LENGTH OF INSTRING IS 7.

THE VALUE OF NEEDLE - 1.

THE VALUE OF CURSOR - 1.

STATUS OF CPOS:

STATUS OF ASTR:

SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.

STR - A

THE VALUE OF NEEDLE - 3.

THE VALUE OF CURSOR - 1.

```
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSPAN.
MSPAN MATCHED SUBSTRING X.
MSPAN MATCHED SUBSTRING Y.
MSPAN MATCHED SUBSTRING X.
MSPAN MATCHED SUBSTRING Y.
MSPAN MATCHED SUBSTRING X.
MSPAN MATCHED SUBSTRING Y.
FIRST - 1
LAST - 7
SUBROUTINE MATCH MATCHED SUBSTRING 'XYXYXY'.
PATTERN MATCHING SUCCEEDED.
```

With tracing facility turned on, an error in the program will be very easily spotted.

4.3 Implementation of the program:

Because of the structure of the program, additional SNOBOL4 features can be added without difficulty. For example, if another primitive function is desired, the size of PATTAB can be increased to include the new feature. A subroutine performing the function must be added to MATCH.

For implementing operators such as the value assignment operators, additional cases would be required in PARSER to recognize the operator and a subroutine performing the assignment must be added to PARSER. This would require more

work than adding pattern matching features but could be accomplished without too much effort.

4.4 Testing of the Program

The program has been tested extensively using different sets of data to make sure all cases and subroutines have been covered and work correctly. An extra long pattern was used and executed without any error. A selected group of test data and the result of their execution are included in Appendix B.

Chapter 5 Summary

The major concern of this project was the design and implementation of a program which performs pattern matching of a subset of SNOBOL4 patterns. The following tasks have been accomplished during the development of this project:

1. Familiarization with SNOBOL4 pattern matching operations

2. Design of the basic data structures

3. Design of the parsing and pattern matching modules

4. Coding

5. Testing

6. Documentation

Software tools played an important role in designing the modules. Structured programming techniques were used to make coding and modification less complicated. A flag bit controls the tracing facility in the program. When it is turned on, the program prints out the values of the variables at each stage of execution. The tracing facility has greatly aided the debugging of the program.

The program has been tested extensively and proven to work correctly for a subset of SNOBOL4 patterns covered in this project. A source listing of the program is included in Appendix A.

REFERENCE

1. Griswold, R. E., J. F. Poage and I. P. Polonsky. The SNOBOL4 Programming Language, Second Edition, Prentice-Hall, Inc., Englewood Cliffs, 1971.

2. Griswold, R. E., J. F. Poage and J. P. Polonsky. The SNOBOL4 Programming Language, Second Edition, pp. 26-29, Prentice-Hall, Inc., Englewood Cliffs, 1971.

3. Griswold, R. E. The Macro Implementation of SNOBOL4, W.H. Freeman and Company, San Francisco, 1972.

4. Gimpel, J. F. The Theory and implementation of Pattern Matching in SNOBOL4 and Other Programming Languages, Bell Telephone Laboratories, Incorporated, Holmdel, 1971.

Appendix A

```
MAIN2:  PROCEDURE OPTIONS(MAIN);
/*****************************************************************/
/*                                                               */
/* PROGRAMMER:  JOSEPHINE LIU                                    */
/* DATE:  FEBRUARY 25, 1977                                      */
/* ABSTRACT:  THIS PROGRAM PERFORMS PARSING AND PATTERN MATCHING */
/* OF A SUBSET OF SNOBOL4 PATTERNS.  IT READS IN A SNOBOL4 PATTERN, */
/* EVALUATES IT AND CREATES A PATTERN STRUCTURE.  THEN USING THE */
/* PATTERN STRUCTURE, IT PERFORMS PATTERN MATCHING ON A SUBJECT  */
/* STRING.                                                       */
/*      THE GENERAL FLOW OF THE PROGRAM STARTS FROM MAIN2.  MAIN2 */
/* INVOKES INITIA TO INITIALIZE THE GLOBAL VARIABLES, AND THEN   */
/* INVOKES PARSER TO RECOGNIZE THE FUNCTION AND OPERATORS OF SNOBOL4*/
/* AND CREATES THE DATA STRUCTURE PATLINK.  AFTER PARSER RETURNS TO */
/* MAIN2 WITH NO ERROR, MAIN2 WILL CALL MATCH TO PERFORM PATTERN */
/* MATCHING ON A SUBJECT STRING.  IF ERROR OCCURS IN EITHER PARSER */
/* OR MATCH, RETURN CODE IS SET TO A VALUE OTHER THAN 0.  THE    */
/* PROGRAM PRINTS AN ERROR MESSAGE FOR THE NON-ZERO RETURN CODE, AND*/
/* GOES ON TO THE NEXT PARSING AND PATTERN MATCHING TASK UNTIL AN */
/* END-OF-FILE CONDITION REACHED.                                */
/*      THERE IS NO PARAMETER PASSING FOR THE INVOCATION OF SUB- */
/* ROUTINES.  ALL THE VARIABLES ARE USED ACCORDING TO THEIR SCOPE. */
/*                                                               */
/*****************************************************************/
/*                                                               */
/* MAIN2 IS A DRIVER WHICH DECLARES ALL GLOBAL VARIABLES, INVOKES */
/* SUBROUTINE INITIA, PARSER AND MATCH TO INITIALIZE VARIABLES,  */
/* PERFORM PARSING AND PATTERN MATCHING, RESPECTIVELY, AND DOES THE */
/* INPUT AND OUTPUT.                                             */
/*                                                               */
/* SUBROUTINES CALLED BY MAIN2 - INITIA, PARSER, MATCH           */
/*                                                               */
```

```
/* THE GLOBAL VARIABLES ARE:                                          */
/* PATTERN - A CONTROLLED VARIABLE ARRAY OF N ELEMENTS.  EACH         */
/*           ELEMENT HAS THE ATTRIBUTE CHARACTER(1).  PATTERN IS      */
/*           USED TO HOLD A SNOBOL4 PATTERN.  THE VALUE OF N DEPENDS  */
/*           UPON THE SIZE OF THE PATTERN.                            */
/* PATLINK - A CONTROLLED VARIABLE HAVING THE FOLLOWING STRUCTURE     */
/*           NUM - CONTAINS THE LOGICAL ADDRESS OF THE PATTERN        */
/*                 COMPONENT AND HAS THE ATTRIBUTE FIXED(2)           */
/*           PAT - CONTAINS THE NAME OF THE COMPONENT AND HAS THE     */
/*                 ATTRIBUTE FIXED(2)                                 */
/*           CAT - HOLDS THE LOGICAL ADDRESS OF THE SUBSEQUENT        */
/*                 COMPONENT TO BE MATCHED AND HAS THE ATTRIBUTE      */
/*                 FIXED(2)                                           */
/*           ALT - HOLDS THE LOGICAL ADDRESS OF THE FIRST COMPONENT   */
/*                 OF THE NEXT ALTERNATE PATTERN AND HAS THE          */
/*                 ATTRIBUTE FIXED(2)                                 */
/*           ARG - IS A POINTER TO THE STARTING POSITION OF THE       */
/*                 PATTERN WHICH ARE STORED IN THE ARGLIST AND HAS    */
/*                 THE ATTRIBUTE FIXED(2)                             */
/*           LEN - CONTAINS THE LENGTH OF THE STRING ARGUMENT         */
/*                 OF THE PATTERN STORED IN ARGLIST, AND HAS THE      */
/*                 ATTRIBUTE FIXED(2)                                 */
/* ARGLIST - AN ARRAY OF 80 ELEMENTS WHICH STORES THE CHARACTERS      */
/*           FROM STRING PATTERN AND ARGUMENT LISTS OF PRIMITIVE      */
/*           FUNCTION.  EACH ELEMENT IN ARGLIST HAS THE ATTRIBUTE     */
/*           CHARACTER(1)                                             */
/* LBV - A CONTROLLED VARIABLE ARRAY.  EACH ELEMENT HAS THE           */
/*       ATTRIBUTE CHARACTER(7) AND CONTAINS THE LABEL OF THE         */
/*       STATEMENT IN THE PROGRAM THAT THE CORRESPONDING COMPONENT    */
/*       IN PATLINK WILL BRANCH TO IN ORDER TO INITIATE THE CORRECT   */
/*       FUNCTION CALL                                                */
/* FLAG1 - A FLAG BIT HAVING THE ATTRIBUTE BIT(1)                     */
/* FLAG2 - A FLAG BIT HAVING THE ATTRIBUTE BIT(1).  IT CONTROLS THE   */
/*         TRACING FACILITY OF THE PROGRAM                            */
```

```
/* ARGPTR - A POINTER TO ARGLIST WITH ATTRIBUTE FIXED(2)                */
/* PATLINKCTR - A POINTER INTO PATLINK HAVING ATTRIBUTE FIXED(2)        */
/* INSTRING - A STRING VARIABLE HOLDING THE SUBJECT STRING TO BE        */
/*            MATCHED AGAINST THE PATTERN, HAVING ATTRIBUTE CHAR(50)    */
/* FIRST - A POINTER TO THE FIRST CHARACTER IN THE SUBSTRING OF THE      */
/*         SUBJECT STRING MATCHED SUCCESSFULLY BY THE PATTERN           */
/*         MATCHING OPERATION                                           */
/* LAST - A POINTER TO THE LAST CHARACTER IN THE SUBSTRING OF THE        */
/*        SUBJECT STRING MATCHED                                        */
/* EOF - A FLAG BIT WITH INITIAL VALUE '0'B FOR CONTINUATION.  IF AN    */
/*       END-OF-FILE CONDITION OCCURS, ITS VALUE WILL BE SET TO '1'B    */
/*       AND THE PROGRAM WILL BE TERMINATED                             */
/* RCODE - A RETURN CODE, WITH INITIAL VALUE 0 FOR NORMAL CONTINUA-     */
/*         TION OF THE PROGRAM.  ANY VALUE OTHER THAN 0 WILL CAUSE      */
/*         AN ERROR MESSAGE TO BE PRINTED                               */
/*                                                                      */
/************************************************************************/
       DCL PATTERN(N) CHARACTER(1) CONTROLLED;
       DCL 1 PATLINK(M) CONTROLLED,
             2 NUM FIXED(2),
             2 PAT CHARACTER(6) VARYING,
             2 CAT FIXED(2),
             2 ALT FIXED(2),
             2 ARG FIXED(2),
             2 LEN FIXED(2);
       DCL (FLAG1,FLAG2) BIT(1);
       DCL LBV(M) CHARACTER(7) CONTROLLED;
       DCL ARGLIST(80) CHARACTER(1);
       DCL ARGPTR FIXED(2);
       DCL PATLINKCTR FIXED(2);
       DCL RCODE FIXED(2);
       DCL MATCHED CHAR(50) VARYING;
       DCL INSTRING CHARACTER(50) VARYING;
       DCL (FIRST,LAST) FIXED(2);
       DCL EOF BIT(1) INITIAL('0'B);
```

```
DCL 1 PATTAB(3),
      2 PRIM CHAR(5) VARYING,
      2 ARG BIT(1),
      2 LBV1 CHAR(6) VARYING;
PATTAB(1).PRIM='LEN';
PATTAB(1).ARG='1'B;
PATTAB(1).LBV1='FLEN';
PATTAB(2).PRIM='SPAN';
PATTAB(2).ARG='1'B;
PATTAB(2).LBV1='FSPAN';
PATTAB(3).PRIM='BREAK';
PATTAB(3).ARG='1'B;
PATTAB(3).LBV1='FBREAK';
ON ENDFILE(SYSIN) EOF='1'B;
DO WHILE (EOF='0'B);
GET EDIT (FLAG2)(COL(1),B(1));
IF EOF='1'B THEN RETURN;
GET EDIT (N) (COL(1),F(3));
ALLOCATE PATTERN(N);
GET EDIT ((PATTERN(I) DO I=1 TO N))(A(1));
PUT PAGE EDIT ('THE PATTERN TO BE USED IS:')(A);
PUT SKIP(0) EDIT ('_____')(A);
PUT SKIP(3) EDIT ((PATTERN(I) DO I=1 TO N))(A(1));
GET EDIT (M) (COL(1),F(2));
ALLOCATE PATLINK(M);
ALLOCATE LBV(M);
CALL INITIA;
CALL PARSER;
IF RCODE=0 THEN FLAG1='1'B;
```

```
DO WHILE (FLAG1='1'B);
  PUT SKIP(3) EDIT ('_____')(COL(20),A);
  DO I=1 TO PATLINKCTR-1;
    PUT SKIP EDIT ('|','|','|')(COL(20),A,COL(30),A,COL(40),A);
    PUT SKIP EDIT ('|',PATLINK(I).NUM,'|',PATLINK(I).PAT
                ,'|')(COL(20),A,COL(25),F(2),COL(30),A,COL(33),
                A,COL(40),A);
    PUT SKIP EDIT ('|','|','|')(COL(20),A,COL(30),A,COL(40),A);
    PUT SKIP(0) EDIT ('_____')(COL(20),A);
    PUT SKIP EDIT ('|','|')(COL(20),A,COL(40),A);
    PUT SKIP EDIT ('|',LBV(I),'|')
                (COL(20),A,COL(27),A,COL(40),A);
    PUT SKIP EDIT ('|','|')(COL(20),A,COL(40),A);
    PUT SKIP(0) EDIT ('_____')(COL(20),A);
    PUT SKIP EDIT ('|','|','|')(COL(20),A,COL(30),A,COL(40),A);
    PUT SKIP EDIT ('|',PATLINK(I).CAT,'|',
                PATLINK(I).ALT,'|')(COL(20),A,COL(25),F(2),
                COL(30),A,COL(35),F(2),COL(40),A);
    PUT SKIP EDIT ('|','|','|')(COL(20),A,COL(30),A,COL(40),A);
    PUT SKIP(0) EDIT ('_____')(COL(20),A);
    PUT SKIP EDIT ('|','|','|')(COL(20),A,COL(30),
                A,COL(40),A);
    PUT SKIP EDIT ('|',PATLINK(I).ARG,'|',PATLINK(I).LEN,'|')
                (COL(20),A,COL(25),F(2),
                COL(30),A,COL(35),F(2),COL(40),A);
    PUT SKIP EDIT ('|','|','|')(COL(20),A,COL(30),
                A,COL(40),A);
    PUT SKIP(0) EDIT('_____')(COL(20),A);
  END;
  FLAG1='0'B;
END;
```

```
        IF RCODE¬=0 THEN DO;
          IF RCODE=1 THEN PUT SKIP EDIT ('ILLEGAL FUNCTION.')(A);
          IF RCODE=2 THEN PUT SKIP EDIT
            ('NO BLANKS BETWEEN FUNCTION & OPERATOR.')(A);
          IF RCODE=3 THEN PUT SKIP EDIT
             ('UNMATCHED PARENTHESIS PAIR.')(A);
          IF RCODE=4 THEN PUT SKIP EDIT ('ILLEGAL OPERATOR.')(A);
          IF RCODE=5 THEN PUT SKIP EDIT ('NO ARGUMENT LIST.')(A);
        END;
        ELSE DO;
INPUT:    GET LIST (INSTRING);
          DO WHILE (INSTRING¬='END');
            FLAG1='0'B;
            CALL MATCH;
            IF RCODE=0 THEN FLAG1='1'B;
            IF FLAG1='1'B THEN DO:
              PUT SKIP(2) EDIT ('FIRST - ',FIRST)(A,F(2));
              PUT SKIP(2) EDIT ('LAST - ',LAST)(A,F(2));
              MATCHED=SUBSTR(INSTRING,FIRST,LAST-FIRST);
              PUT SKIP(2) EDIT
                ('SUBROUTINE MATCH MATCHED SUBSTRING ',
                '''',MATCHED,'''','.')
                (A,A,A,A,A);
            END;
            IF RCODE¬=0 THEN PUT SKIP(2) EDIT
              ('PATTERN MATCHING FAILED.')(A);
            ELSE PUT SKIP(2) EDIT ('PATTERN MATCHING SUCCEEDED.')
              (A);
            GO TO INPUT;
          END;
        END;
```

```
        FREE PATTERN;
        FREE PATLINK;
        FREE LBV;
        END; /* END WHILE EOF='0'B */
/*********************************************************************/
/*                                                                   */
/* INITIA INITIALIZES THE GLOBAL VARIABLES DECLARED IN MAIN2.        */
/*                                                                   */
/* INVOKING ROUTINE - MAIN2                                          */
/*                                                                   */
/*********************************************************************/
INITIA: PROCEDURE;
        DO I=1 TO M;
          CAT(I)=0;
          ALT(I)=0;
          LEN(I)=0;
        END;
        FLAG1='0'B;
        ARGPTR=1;
        PATLINKCTR=1;
        RCODE=0;
        END INITIA;
/*********************************************************************/
/*                                                                   */
/* PARSER USE THE INDEX VARIABLE PTR TO SCAN THROUGH THE ARRAY       */
/* PATTERN AND TO RECOGNIZE EACH PATTERN COMPONENT AND OPERATOR, AND */
/* FILL IN THE DATA STRUCTURE PATLINK.                               */
/*                                                                   */
/* INVOKING ROUTINE - MAIN2                                          */
/* VARIABLES PASSED FROM MAIN2 - PATTERN, PATLIK, LBV, PATLINKCTR    */
/*                               ARGLIST, ARGPTR, RCODE              */
/*                                                                   */
/* SUBROUTINES INVOKED BY PARSER - STRING, PATREC                    */
/*                                                                   */
```

```
/* THE VARIABLES LOCAL TO PARSER ARE:                                  */
/* CSTK - A STACK HOLDING THE LOGICAL ADDRESS OF COMPONENTS WAITING */
/*        FOR A CONCATENATION OPERATOR                                 */
/* ASTK - A STACK HOLDING THE LOGICAL ADDRESS OF COMPONENTS WAITING */
/*        FOR AN ALTERNATION OPERATOR                                  */
/* CURTOK - A VARIABLE USED BY PARSER TO TEMPORARILY STORE THE NAME */
/*          OF A PRIMITIVE FUNCTION WITH ATTRIBUTE CHARACTER(5)        */
/* PTR - A POINTER TO THE ARRAY PATTERN AND IS USED BY PARSER TO    */
/*       SCAN THE PATTERN.  IT HAS ATTRIBUTE FIXED(3)                 */
/* A98 - A COUNTER OF LEFT PARENTHESIS IN ASTK                       */
/* A99 - A COUNTER OF RIGHT PARENTHESIS IN ASTK                      */
/* C98 - A COUNTER OF LEFT PARENTHESIS IN CSTK                       */
/* C99 - A COUNTER OF RIGHT PARENTHESIS IN CSTK                      */
/* A - A POINTER TO ASTK                                             */
/* C - A POINTER TO CSTK                                             */
/*                                                                   */
/*********************************************************************/
PARSER: PROCEDURE;
        /* '('=98 */
        /* ')'=99 */
        DCL (CSTK,ASTK)(M+4) FIXED(2) INIT((M+4)(0)),
            CURTOK CHARACTER(6) VARYING INIT(''),
            PTR FIXED(3) INIT(1),
            PCTR FIXED(2),
            (A98,A99,C98,C99) FIXED(2) INIT(0),
            (A,C) FIXED(2) INIT(0);
        DO WHILE (PTR<=N);
/*********************************************************************/
/*                                                                   */
/*  CASE 1 - A LEFT PARENTHESIS IS POINTED TO BY PTR.  PARSER WILL   */
/*  PUSH THE VALUE 98 ON BOTH ASTK AND CSTK.                         */
/*                                                                   */
/*********************************************************************/
```

```
          IF PATTERN(PTR)='(' THEN DO;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
              ('SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.')
              (A);
            A=A+1;
            ASTK(A)=98;
            C=C+1;
            CSTK(C)=98;
            PTR=PTR+1;
            DO WHILE (PATTERN(PTR)=' ');
              PTR=PTR+1;
            END;
            IF PATTERN(PTR)¬='(' THEN DO;
              IF PATTERN(PTR)='''' THEN DO;
                PTR=PTR+1;
                CALL STRING;
              END;
              ELSE CALL PATREC;
              IF RCODE¬=0 THEN RETURN;
              A=A+1;
              ASTK(A)=PATLINKCTR;
              C=C+1;
              CSTK(C)=PATLINKCTR;
              PATLINKCTR=PATLINKCTR+1;
            END;
          END;
/*************************************************************************/
/*              END CASE 1                                               */
```

```
/*****************************************************************/
/*                                                               */
/*  CASE 2 - A QUOTE IS POINTED TO BY PTR.  PARSER WILL CALL     */
/*  SUBROUTINE STRING.  UPON RETURNING FROM STRING, PARSER WILL PUSH*/
/*  THE CURRENT VALUE OF PATLINKCTR ON BOTH ASTK AND CSTK AND    */
/*  INCREMENT PATLINKCTR BY 1.                                   */
/*                                                               */
/*****************************************************************/

          ELSE IF PATTERN(PTR)='''' THEN DO;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
              ('SUBROUTINE PARSER ENTERING CASE 2.')(A);
            PTR=PTR+1;
            CALL STRING;
            A=A+1;
            ASTK(A)=PATLINKCTR;
            C=C+1;
            CSTK(C)=PATLINKCTR;
            PATLINKCTR=PATLINKCTR+1;
          END;
/*****************************************************************/
/*              END CASE 2                                       */
/*****************************************************************/
/*                                                               */
/*  CASE 3 - A RIGHT PARENTHESIS IS POINTED TO BY PTR.  PARSER WILL */
/*  LOOK AHEAD FOR ANY ONE OF THE FOLLOWING FOUR POSSIBILITIES AND */
/*  PERFORM THE SUBSEQUENT ACTIONS:                              */
/*  1.   END OF PATTERN - PARSER WILL PUSH THE RIGHT PARENTHESIS */
/*       (99) ON BOTH ASTK AND CSTK.                             */
/*  2.   ANOTHER RIGHT PARENTHESIS - PARSER WILL POP EVERYTHING UP */
/*       TO AND INCLUDING A LEFT PARENTHESIS FROM ASTK, THEN PUSH */
/*       THE RIGHT PARENTHESIS ON CSTK.                          */
```

```
/*  3.    AN ALTERNATION OPERATOR - PARSER WILL PUSH THE RIGHT       */
/*        PARENTHESIS ON CSTK, THEN POP EVERYTHING OUT OF ASTK UP TO */
/*        AND INCLUDING A LEFT PARENTHESIS.  IF ASTK IS NOT EMPTY,   */
/*        PARSER WILL PUT THE CURRENT VALUE OF PATLINKCTR IN THE ALT */
/*        FIELD OF THE PATTERN COMPONENT ON TOP OF ASTK AND POP THE  */
/*        TOP OF ASTK.                                               */
/*  4.    A CONCATENATION OPERATOR - PARSER WILL POP EVERYTHING UP TO*/
/*        AND INCLUDING A LEFT PARENTHESIS FROM ASTK AND PUT THE     */
/*        CURRENT VALUE OF PATLINKCTR IN THE CAT FIELD OF ALL THE    */
/*        COMPONENTS IN CSTK UP TO A LEFT PARENTHESIS.               */
/*                                                                   */
/*********************************************************************/
          ELSE IF PATTERN(PTR)=')' THEN DO;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
              ('SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.')
              (A);
            PTR=PTR+1;
            I=0;     /* COUNT FOR BLANKS */
            IF PTR<=N THEN DO;
              DO WHILE (PATTERN(PTR)=' ');/* BLANKS MUST BE PRESENT */
                I=I+1;                   /* BETWEEN FUNCTION AND OPERATOR */
                PTR=PTR+1;
              END;
              IF PATTERN(PTR)¬=')' THEN DO;
                IF I=0 THEN DO;
                  RCODE=2;
                  RETURN;
                END;
                IF PATTERN(PTR)='|' THEN DO;
                  C=C+1;
                  CSTK(C)=99;
```

```
    DO WHILE (A>1&ASTK(A)¬=98);
      A=A-1;
    END;
    IF ASTK(A)¬=98|A<=0 THEN DO;
      RCODE=3;
      RETURN;
    END;
    ELSE IF A=1&ASTK(A)=98 THEN DO;
      A=A-1;
      PTR=PTR+1;
    END;
    ELSE DO;
      A=A-1;
      PATLINK(ASTK(A)).ALT=PATLINKCTR;
      PTR=PTR+1;
    END;
    I=0;
    DO WHILE (PATTERN(PTR)=' ');
      I=I+1;
      PTR=PTR+1;
    END;
    IF I=0 THEN DO;
      RCODE=2;
      RETURN;
    END;
  END;
  ELSE DO;
    DO WHILE (ASTK(A)¬=98);
      A=A-1;
      IF A<1 THEN DO;
        RCODE=3;
        RETURN;
      END;
```

```
        END;
        A=A-1;
        PCTR=0;      /* COUNT EACH ')' POPPED */
        DO WHILE (CSTK(C)¬=98);
          IF CSTK(C)=99 THEN DO;
            PCTR=PCTR+1;
            C=C-1;
          END;
          PATLINK(CSTK(C)).CAT=PATLINKCTR;
          C=C-1;
          IF PCTR>0&CSTK(C)=98 THEN DO;
            PCTR=PCTR-1;
            C=C-1;/* POP '(' THAT MATCHED EACH ')' POPPED */
          END;
        END;
        C=C-1;
        IF PCTR¬=0 THEN DO;
          RCODE=3;
          RETURN;
        END;
      END;
    END;
    ELSE DO;
      DO WHILE (ASTK(A)¬=98);
        A=A-1;
      END;
      A=A-1;
      C=C+1;
      CSTK(C)=99;
    END;
  END;
```

```
            ELSE DO;
              A=A+1;
              ASTK(A)=99;
              C=C+1;
              CSTK(C)=99;
            END;
          END;
/*****************************************************************/
/*               END CASE 3                                      */
/*****************************************************************/
/*                                                               */
/*  CASE 4 - A PRIMITIVE FUNCTION IS POINTED TO BY PTR.  PARSER*/
/*  WILL CALL PATREC TO RECOGNIZE THE PRIMITIVE FUNCTION AND   */
/*  FILL IN PATLINK.                                           */
/*                                                               */
/*****************************************************************/
          ELSE DO;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
              ('SUBROUTINE PARSER ENTERING CASE 4.')(A);
            CALL PATREC;
            IF RCODE¬=0 THEN RETURN;
            A=A+1;
            ASTK(A)=PATLINKCTR;
            C=C+1;
            CSTK(C)=PATLINKCTR;
            PATLINKCTR=PATLINKCTR+1;
          END;
/*****************************************************************/
/*               END CASE 4                                      */
```

```
/*****************************************************************/
/*                                                               */
/*  CASE 5 - AN OPERATOR, EITHER CONCATENATION OR ALTERNATION    */
/*  IS POINTED TO BY PTR.  IF IT IS A CONCATENATION OPERATOR     */
/*  PARSER WILL POP A COMPONENT FROM THE TOP OF CSTK AND LINK    */
/*  THE CAT FIELD OF THAT COMPONENT WITH THE NEXT COMPONENT.     */
/*  IF IT IS AN ALTERNATION OPERATOR, PARSER WILL POP THE        */
/*  FIRST ELEMENT FROM ASTK AND LINKS ITS ALT FIELD WITH THE     */
/*  NEXT COMPONENT.                                              */
/*                                                               */
/*****************************************************************/
          IF PATTERN(PTR)=' '&PTR<N THEN DO;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
              ('SUBROUTINE PARSER ENTERING CASE 5.')(A);
            PTR=PTR+1;
            DO WHILE (PATTERN(PTR)=' ');
              PTR=PTR+1;
            END;
            IF (PATTERN(PTR)=')'|PATTERN(PTR)='''|PATTERN(PTR+1)¬=' ')
              &PATTERN(PTR)¬='|'&PATTERN(PTR)¬=')' THEN DO;
              PATLINK(PATLINKCTR-1).CAT=PATLINKCTR;
              C=C-1;
            END;
            ELSE IF PATTERN(PTR)='|' THEN DO;
              DO WHILE (A>1&ASTK(A-1)¬=98);
                A=A-1;
              END;
              PATLINK(ASTK(A)).ALT=PATLINKCTR;
              PTR=PTR+1;
              A=A-1;
              I=0;
```

```
            DO WHILE (PATTERN(PTR)=' ');
              I=I+1;
              PTR=PTR+1;
            END;
            IF I=0 THEN DO;
              RCODE=2;
              RETURN;
            END;
          END;
          ELSE IF PATTERN(PTR)¬=')' THEN DO;
            RCODE=4;
            RETURN;
          END;
        END;
/*************************************************************************/
/*                    END CASE 5                                         */
/*************************************************************************/
        IF FLAG2='1'B THEN DO;
          PUT SKIP(2) EDIT ('STATUS OF ASTK:')(A);
          PUT EDIT ((ASTK(I) DO I=1 TO A))(COL(8),F(2));
          PUT SKIP(2) EDIT ('STATUS OF CSTK:')(A);
          PUT EDIT ((CSTK(I) DO I=1 TO C))(COL(8),F(2));
          PUT SKIP(2) EDIT ('STATUS OF ARGLIST:')(A);
          PUT EDIT ((ARGLIST(I) DO I=1 TO ARGPTR-1))(COL(8),A(1));
        END;
      END;
      IF A¬=0 THEN DO;
        DO I=1 TO A;
          IF ASTK(I)=98 THEN A98=A98+1;
          IF ASTK(I)=99 THEN A99=A99+1;
        END;
        IF A98¬=A99 THEN DO;
          RCODE=3;
          RETURN;
        END;
      END;
```

```
        IF C¬=0 THEN DO;
          DO I=1 TO C;
            IF CSTK(I)=98 THEN C98=C98+1;
            IF CSTK(I)=99 THEN C99=C99+1;
          END;
          IF C98¬=C99 THEN DO;
            RCODE=3;
            RETURN;
          END;
        END;
/*****************************************************************/
/*                                                               */
/*  STRING IS INVOKED WHENEVER A STRING COMPONENT IS             */
/*  ENCOUNTERED.  IT ADDS THE CHARACTERS BETWEEN A PAIR OF       */
/*  QUOTES TO ARGLIST, FILLS IN THE DATA STRUCTURE PATLINK WITH  */
/*  THE PROPER INFORMATION, AND INCREMENTS THE COUNTER VARIABLE  */
/*  PTR AND ARGPTR, BEFORE RETURNING TO THE CALLING PROCEDURE    */
/*                                                               */
/*  INVOKING ROUTINE - PARSER                                    */
/*  THE VARIABLES PASSED FROM CALLING ROUTINE - PATLINK, PTR,    */
/*  ARGLIST, ARGPTR, PATTERN, LBV                                */
/*                                                               */
/*****************************************************************/
STRING: PROCEDURE;
        IF FLAG2='1'B THEN PUT SKIP(2) EDIT
          ('SUBROUTINE PARSER ENTERING SUBROUTINE STRING.')(A);
        I=0;
        PATLINK(PATLINKCTR).ARG=ARGPTR;
        DO WHILE (PATTERN(PTR)¬='''');
          ARGLIST(ARGPTR)=PATTERN(PTR);
          ARGPTR=ARGPTR+1;
          I=I+1;
          PTR=PTR+1;
        END;
```

```
        PATLINK(PATLINKCTR).NUM=PATLINKCTR;
        PATLINK(PATLINKCTR).PAT='STRING';
        LBV(PATLINKCTR)='FSTRING';
        PATLINK(PATLINKCTR).LEN=I;
        PTR=PTR+1;
        RETURN;
        END STRING;
/*****************************************************************/
/*                                                               */
/*  PATREC IS INVOKED WHEN A POSSIBLE PRIMITIVE FUNCTION IS      */
/*  DETECTED.  IT TAKES THE NAME STORED IN CURTOK, AND PERFORMS  */
/*  A TABLE LOOK-UP IN PATTAB TO MAKE CERTAIN THE PRIMITIVE      */
/*  FUNCTION IS LEGAL.  THEN PATREC FILLS IN PATLINK AND         */
/*  INCREMENTS COUNTER VARIABLES PTR AND ARGPTR, BEFORE          */
/*  RETURNING TO THE CALLING ROUTINE.  IF AN ERROR CONDITION     */
/*  OCCURS, SUCH AS AN ILLEGAL FUNCTION OR NO ARGUMENT LISTS,    */
/*  THE RETURN CODE RCODE WILL BE SET BEFORE RETURN AND NO       */
/*  OTHER ACTIONS WILL BE TAKEN.                                 */
/*                                                               */
/*  INVOKING ROUTINE - PARSER                                    */
/*  THE VARIABLES PASSED BY THE INVOKING ROUTINE - PATTERN       */
/*  CURTOK, PATTAB, PATLINK, PATLINKCTR, PTR, ARGLIST, ARGPTR,   */
/*  RCODE                                                        */
/*                                                               */
/*****************************************************************/
PATREC: PROCEDURE;
        IF FLAG2='1'B THEN PUT SKIP(2) EDIT
          ('SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.')(A);
        DO WHILE (PATTERN(PTR)¬=' '&PATTERN(PTR)¬='(');
          CURTOK=CURTOK||PATTERN(PTR);
          PTR=PTR+1;
        END;
```

```
DO J=1 TO 3;
  IF PATTAB(J).PRIM=CURTOK THEN DO;
    PATLINK(PATLINKCTR).NUM=PATLINKCTR;
    PATLINK(PATLINKCTR).PAT=CURTOK;
    CURTOK='';
    IF PATTAB(J).ARG='1'B THEN DO;
      DO WHILE (PATTERN(PTR)=' ');
        PTR=PTR+1;
      END;
      IF PATTERN(PTR)='(' THEN DO;
        PTR=PTR+1;
        I=0;
        DO WHILE(PATTERN(PTR)¬=')');
          IF PATTERN(PTR)='''' THEN DO;
            PATLINK(PATLINKCTR).ARG=ARGPTR;
            PTR=PTR+1;
            DO WHILE (PATTERN(PTR)¬='''');
              ARGLIST(ARGPTR)=PATTERN(PTR);
              PTR=PTR+1;
              I=I+1;
              ARGPTR=ARGPTR+1;
            END;
            PTR=PTR+1;
          END;
          ELSE DO;
            DO WHILE (PATTERN(PTR)¬=')');
              CURTOK=CURTOK||PATTERN(PTR);
              PTR=PTR+1;
              I=I+1;
            END;
            PATLINK(PATLINKCTR).ARG=CURTOK;
          END;
        END;
```

```
                PATLINK(PATLINKCTR).LEN=I;
                PTR=PTR+1;
              END;
              ELSE DO;
                RCODE=5;
                RETURN;
              END;
            END;
            LBV(PATLINKCTR)=PATTAB(J).LBV1;
            J=4;
          END;
          ELSE IF J>=3 THEN DO;
            RCODE=1;
            RETURN;
          END;
        END;
        CURTOK='';
        RETURN;
        END PATREC;
        RETURN;
        END PARSER;
/*****************************************************************/
/*                                                               */
/*  MATCH PERFORMS THE ACTUAL PATTERN MATCHING FOR THE PROGRAM.  */
/*  IT READS A SUBJECT STRING INTO INSTRING THEN DECIDES WHICH   */
/*  SUBROUTINE IT SHOULD CALL TO PERFORM PROPER PATTERN          */
/*  MATCHING.                                                    */
/*                                                               */
/*  INVOKING ROUTINE - MAIN2                                     */
/*  THE VARIABLES PASSED FROM INVOKING ROUTINE - INSTRING,       */
/*    PATLINK, PATLINKCTR, ARGLIST, LBV, RCODE, LAST, FIRST      */
/*                                                               */
```

```
/*  SUBROUTINES CALLED BY MATCH - MSTRING, MSPAN, MBREAK, MLEN */
/*  THE VARIABLES LOCAL TO MATCH ARE:                          */
/*  CURSOR - POINTING TO THE POSITION IN THE SUBJECT STRING    */
/*           WHERE THE PATTERN MATCHING ROUTINE IS SEARCHING   */
/*           FOR A MATCH TO THE CURRENT PATTERN COMPONENT      */
/*  NEEDLE - A POINTER TO THE CURRENT COMPONENT OF THE PATTERN */
/*           STRUCTURE PATLINK                                 */
/*  CPOS - HOLDS THE NUMBER OF POSITIONS CURSOR HAS BEEN       */
/*         ADVANCED IN EACH SUCCESSFUL MATCH                   */
/*  ASTK - CONTAINS THE ALT FIELDS OF THE CURRENT SUCCESSFULLY */
/*         MATCHED COMPONENTS                                  */
/*  APTR - POINTER TO THE ASTK                                 */
/*  CUPTR - POINTER TO THE CPOS                                */
/*  BINGO - A FLAG BIT TO MARK THE FIRST SUCCESSFUL MATCH      */
/*                                                             */
/*************************************************************/
MATCH:  PROCEDURE;
        DCL CURSOR FIXED(2) INIT(1),
            NEEDLE FIXED(2) INIT(1),
            (APTR,CUPTR) FIXED(2) INIT(0),
            BINGO BIT(1) INIT('0'B),
            L FIXED(2),
            (CPOS,ASTK)(10) FIXED(2) INITIAL ((10)(0));
        PUT SKIP(5) EDIT ('THE STRING TO BE MATCHED IS:  ',
                          '''',INSTRING,'''','.')(A,A,A,A,A);
        I=LENGTH(INSTRING);
        IF FLAG2='1'B THEN
          PUT SKIP(2) EDIT ('THE LENGTH OF INSTRING IS ',I,'.')
                           (A,F(2),A);
```

```
        DO WHILE (CURSOR<=I&NEEDLE<=PATLINKCTR-1);
          IF FLAG2='1'B THEN DO;
            PUT SKIP(2) EDIT
            ('THE VALUE OF NEEDLE - ',NEEDLE,'.') (A,F(2),A);
            PUT SKIP(2) EDIT
            ('THE VALUE OF CURSOR - ',CURSOR,'.') (A,F(2),A);
            PUT SKIP(2) EDIT ('STATUS OF CPOS:') (A);
            PUT EDIT ((CPOS(K) DO K=1 TO CUPTR)) (COL(8),F(2));
            PUT SKIP(2) EDIT ('STATUS OF ASTK:') (A);
            PUT EDIT ((ASTK(K) DO K=1 TO APTR)) (COL(8),F(2));
          END;
          IF LBV(NEEDLE)='FSTRING' THEN GO TO FSTRING;
          IF LBV(NEEDLE)='FLEN' THEN GO TO FLEN;
          IF LBV(NEEDLE)='FSPAN' THEN GO TO FSPAN;
          IF LBV(NEEDLE)='FBREAK' THEN GO TO FBREAK;
          ELSE PUT SKIP(2) EDIT ('ERROR IN LINKED STRUCTURE.')
                                (A);
          RETURN;
FSTRING:  CALL MSTRING;
          IF RCODE¬=0 THEN GO TO BACKUP;
          ELSE DO;
            IF BINGO='0' THEN DO;
              FIRST=CURSOR-PATLINK(NEEDLE).LEN;
              BINGO='1'B;
            END;
            GO TO SUCCESS;
          END;
FSPAN:    CALL MSPAN;
          IF RCODE¬=0 THEN GO TO BACKUP;
          ELSE DO;
            IF BINGO='0'B THEN DO;
              FIRST=L;
              BINGO='1'B;
            END;
            GO TO SUCCESS;
          END;
```

```
FBREAK:   CALL MBREAK;
          IF RCODE¬=0 THEN GO TO BACKUP;
          ELSE DO;
            IF BINGO='0'B THEN DO;
              FIRST=CURSOR-L;
              BINGO='1'B;
            END;
            GO TO SUCCESS;
          END;
FLEN:     CALL MLEN;
          IF RCODE¬=0 THEN GO TO BACKUP;
          ELSE DO;
            IF BINGO='0'B THEN DO;
              FIRST=CURSOR-PATLINK(NEEDLE).ARG;
              BINGO='1'B;
            END;
            GO TO SUCCESS;
          END;
BACKUP:   IF PATLINK(NEEDLE).ALT¬=0 THEN DO;
            RCODE=0;
            NEEDLE=PATLINK(NEEDLE).ALT;
          END;
          ELSE IF BINGO='0'B THEN DO;
            RCODE=0;
            NEEDLE=1;
            CURSOR=CURSOR+1;
          END;
          ELSE IF APTR>=1 THEN DO;
            RCODE=0;
            NEEDLE=ASTK(APTR);
            APTR=APTR-1;
            CURSOR=CURSOR-CPOS(CUPTR);
            CUPTR=CUPTR-1;
          END;
```

```
            ELSE RETURN;
            GO TO AGAIN;
SUCCESS:    IF PATLINK(NEEDLE).CAT=0 THEN DO;
              LAST=CURSOR;
              RETURN;
            END;
            ELSE DO;
              IF PATLINK(NEEDLE).ALT¬=0 THEN DO;
                APTR=APTR+1;
                ASTK(APTR)=PATLINK(NEEDLE).ALT;
              END;
              NEEDLE=PATLINK(NEEDLE).CAT;
            END;
AGAIN:    END;
/*****************************************************************/
/*                                                               */
/*  MSTRING DOES THE ACTUAL PATTERN MATCHING FOR  A STRING PATTERN.  */
/*  IT USES THE VALUE IN THE ARG FIELD OF THE PATLINK ELEMENT THAT   */
/*  NEEDLE IS CURRENTLY POINTING TO AS THE BEGINNING POINTER TO      */
/*  ARGLIST.  IT THEN EXTRACTS THE NUMBER OF CHARACTERS SPECIFIED IN */
/*  THE LEN FIELD FROM ARGLIST.  FINALLY, IT COMPARES THE STRING FROM*/
/*  ARGLIST WITH THE SUBSTRING OF THE SUBJECT STRING POINTED TO BY   */
/*  CURSOR.  IF THE SUBSTRING SUCCESSFULLY MATCHES THE STRING PATTERN*/
/*  CURSOR WILL BE ADVANCED TO THE RIGHT OF THE LAST CHARACTER IN THE*/
/*  SUBSTRING MATCHED.                                               */
/*                                                                   */
/*  INVOKING ROUTINE - MATCH                                         */
/*  THE VARIABLES PASSED FROM CALLING ROUTINE - PATLINK, ARGLIST,    */
/*           CURSOR, NEEDLE, CPOS, CUPTR, RCODE                      */
/*                                                                   */
```

```
/*   THE VARIABLES LOCAL TO MSTRING:                                    */
/*   KTR - A COUNTER VARIABLE                                           */
/*   STR - A CHARACTER STRING HOLDING THE ARGUMENT OF STRING PATTERN    */
/*   J - A COUNTER VARIABLE                                             */
/*                                                                      */
/************************************************************************/
MSTRING:  PROCEDURE;
          DCL KTR FIXED(2),
              STR CHARACTER(30) VARYING INIT(''),
              J FIXED(2);
          IF FLAG2='1'B THEN PUT SKIP(2) EDIT
            ('SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.')
            (A);
          J=PATLINK(NEEDLE).ARG+PATLINK(NEEDLE).LEN-1;
          DO KTR=PATLINK(NEEDLE).ARG TO J;
            STR=STR||ARGLIST(KTR);
          END;
          IF FLAG2='1'B THEN PUT SKIP(2) EDIT ('STR - ',STR)
                                              (A,A);
          IF STR=SUBSTR(INSTRING,CURSOR,PATLINK(NEEDLE).LEN)
            THEN DO;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
              ('MSTRING MATCHED SUBSTRING ','''',STR,'''','.')
              (A,A,A,A,A);
            IF PATLINK(NEEDLE).ALT¬=0 THEN DO;
              CUPTR=CUPTR+1;
              CPOS(CUPTR)=PATLINK(NEEDLE).LEN;
            END;
            CURSOR=CURSOR+PATLINK(NEEDLE).LEN;
          END;
          ELSE RCODE=1;
          RETURN;
          END MSTRING;
```

```
/*****************************************************************************/
/*                                                                           */
/*  MSPAN PERFORMS THE ACTUAL PATTERN MATCHING FOR THE PRIMITIVE             */
/*  FUNCTION SPAN.  IT CHECKS EACH CHARACTER IN THE SUBSTRING OF THE         */
/*  SUBJECT STRING POINTED TO BY CURSOR AGAINST THE ARGUMENT STRING          */
/*  STORED IN ARGLIST.  IF THE CURRENT CHARACTER APPEARS IN THE              */
/*  ARGUMENT STRING, CURSOR IS ADVANCED TO THE NEXT CHARACTER IN THE         */
/*  SUBJECT STRING.  THE PROCESS CONTINUES UNTIL A CHARACTER WHICH           */
/*  DOES NOT APPEAR IN THE ARGUMENT STRING IS REACHED OR THE END OF          */
/*  THE SUBJECT STRING IS REACHED.                                           */
/*                                                                           */
/*  INVOKING ROUTINE - MATCH                                                 */
/*  THE VARIABLES PASSED FROM CALLING ROUTINE - PATLINK, ARGLIST,            */
/*     INSTRING, NEEDLE, CURSOR, CPOS, CUPTR, RCODE                          */
/*                                                                           */
/*  THE VARIABLES LOCAL TO MSPAN ARE:                                        */
/*  SWITCH - A SWITCH WHICH WILL BE TURNED ON WHEN A SPAN CHARACTER          */
/*           IS NOT FOUND                                                    */
/*  KTR - A COUNTER VARIABLE                                                 */
/*  J - A COUNTER VARIABLE                                                   */
/*                                                                           */
/*****************************************************************************/
MSPAN:    PROCEDURE;
          DCL SWITCH FIXED(1) INIT(0),
              KTR FIXED(2),
              J FIXED(2);
          IF FLAG2='1'B THEN PUT SKIP(2) EDIT
           ('SUBROUTINE MATCH ENTERING SUBROUTINE MSPAN.')
           (A);
          L=CURSOR;
```

```
        DO WHILE (SWITCH=0&CURSOR<=I);
          J=PATLINK(NEEDLE).ARG+PATLINK(NEEDLE).LEN-1;
          DO KTR=PATLINK(NEEDLE).ARG TO J;
            IF SUBSTR(INSTRING,CURSOR,1)=ARGLIST(KTR) THEN DO;
              IF FLAG2='1'B THEN PUT SKIP(2) EDIT
                ('MSPAN MATCHED SUBSTRING ',ARGLIST(KTR),'.')
                (A,A,A);
              CURSOR=CURSOR+1;
              J=J+1;
            END;
            ELSE IF KTR=J THEN SWITCH=1;
          END;
        END;
        IF L=CURSOR THEN DO;
          RCODE=1;
          RETURN;
        END;
        ELSE IF PATLINK(NEEDLE).ALT¬=0 THEN DO;
          CUPTR=CUPTR+1;
          CPOS(CUPTR)=CURSOR-L;
        END;
        RETURN;
        END MSPAN;
/*****************************************************************************/
/*                                                                           */
/*  MBREAK PERFORMS ACTUAL PATTERN MATCHING FOR THE PRIMITIVE                */
/*  FUNCTION BREAK.  IT STARTS AT THE CHARACTER IN THE SUBSTRING OF          */
/*  THE SUBJECT STRING POINTED TO BY CURSOR.  IF THE CHARACTER IS NOT        */
/*  IN THE ARGUMENT STRING, CURSOR WILL BE ADVANCED TO THE NEXT              */
/*  CHARACTER.  THE PROCESS CONTINUES UNTIL A CHARACTER IN THE               */
/*  SUBJECT STRING IS FOUND IN THE ARGUMENT STRING.  PATTERN MATCHING        */
/*  SUCCEEDS WHEN A BREAK CHARACTER IS FOUND.                                */
/*                                                                           */
```

```
/*  INVOKING ROUTINE - MATCH                                              */
/*  THE VARIABLES PASSED FROM CALLING ROUTINE - PATLINK, ARGLIST,         */
/*     INSTRING, NEEDLE, CURSOR, CPOS, CUPTR, RCODE                       */
/*                                                                        */
/*  THE VARIABLES LOCAL TO MBREAK:                                        */
/*  SWTICH - A SWITCH TO BE TURNED ON WHEN A BREAK CHARACTER IS FOUND     */
/*  SKIPCHAR - A CHARACTER POINTED TO BY CURSOR WHICH IS NOT A BREAK      */
/*             CHARACTER                                                  */
/*  KTR AND J - COUNTER VARIABLES                                         */
/*                                                                        */
/**************************************************************************/
MBREAK:   PROCEDURE;
          DCL SWITCH FIXED(1) INIT(0),
              SKIPCHAR CHAR(1),
              (KTR,J) FIXED(2);
          IF FLAG2='1'B THEN PUT SKIP(2) EDIT
            ('SUBROUTINE MATCH ENTERING SUBROUTINE MBREAK.')
            (A);
          L=0;
          J=PATLINK(NEEDLE).ARG+PATLINK(NEEDLE).LEN-1;
          DO WHILE (SWITCH=0&CURSOR<=I);
            DO KTR=PATLINK(NEEDLE).ARG TO J;
              IF SUBSTR(INSTRING,CURSOR,1)=ARGLIST(KTR) THEN DO;
                SWITCH=1;
                KTR=J+1;
              END;
            END;
            IF SWITCH=0 THEN DO;
              SKIPCHAR=SUBSTR(INSTRING,CURSOR,1);
              IF FLAG2='1'B THEN PUT SKIP(2) EDIT
                ('MBREAK SKIPPED SUBSTRING ','''',
                 SKIPCHAR,'''','.')
                (A,A,A,A,A);
              CURSOR=CURSOR+1;
              L=L+1;
            END;
```

```
            ELSE DO;
              SKIPCHAR=SUBSTR(INSTRING,CURSOR,1);
              IF FLAG2='1'B THEN PUT SKIP(2) EDIT
                 ('MBREAK FOUND BREAK CHARACTER ',''''
                  SKIPCHAR,'''','.')
                 (A,A,A,A,A);
              END;
            END;
            IF SWITCH=0 THEN DO;
              PUT SKIP(2) EDIT ('BREAK CHARACTERS NOT FOUND.')(A);
              CURSOR=CURSOR-L;
              RCODE=1;
            END;
            ELSE IF PATLINK(NEEDLE).ALT¬=0 THEN DO;
              CUPTR=CUPTR+1;
              CPOS(CUPTR)=L;
            END;
            RETURN;
            END MBREAK;
/*****************************************************************/
/*                                                               */
/*  MLEN PERFORMS ACTUAL PATTERN MATCHING FOR THE PRIMITIVE FUNCTION */
/*  LEN(N).  IT CHECKS THE LENGTH OF THE SUBJECT STRING FROM THE  */
/*  CHARACTER POINTED TO BY THE CURSOR.  IF THIS SUBSTRING HAS N OR */
/*  MORE CHARACTERS, PATTERN MATCHING SUCCEEDS.                  */
/*                                                               */
/*  INVOKING ROUTINE - MATCH                                     */
/*  THE VARIABLE PASSED BY CALLING ROUTINE - PATLINK, INSTRING,  */
/*     NEEDLE, CPOS, CUPTR, RCODE, CURSOR                        */
/*                                                               */
/*****************************************************************/
MLEN:       PROCEDURE;
            IF FLAG2='1'B THEN PUT SKIP(2) EDIT
```

```
    ('SUBROUTINE MATCH ENTERING SUBROUTINE MLEN.') (A);
  IF (CURSOR+PATLINK(NEEDLE).ARG-1)<=I THEN DO;
    IF FLAG2='1'B THEN PUT SKIP(2) EDIT
      ('MLEN MATCHED SUBSTRING ',''''
      SUBSTR(INSTRING,CURSOR,PATLINK(NEEDLE).ARG),''''.')
      (A,A,A,A,A);
    IF PATLINK(NEEDLE).ALT¬=0 THEN DO;
      CUPTR=CUPTR+1;
      CPOS(CUPTR)=PATLINK(NEEDLE).ARG;
    END;
    CURSOR=CURSOR+PATLINK(NEEDLE).ARG;
  END;
  ELSE RCODE=1;
  RETURN;
  END MLEN;
  END MATCH;
END MAIN2;
```

Appendix B

THE PATTERN TO BE USED IS:

```
('AB' | LEN(4)) BREAK('.')
SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.
SUBROUTINE PARSER ENTERING SUBROUTINE STRING.
SUBROUTINE PARSER ENTERING CASE 5.
STATUS OF ASTK:
        98
STATUS OF CSTK:
        98
         1
STATUS OF ARGLIST:
        A
        B
SUBROUTINE PARSER ENTERING CASE 4.
SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.
STATUS OF ASTK:
        98
         2
STATUS OF CSTK:
        98
         1
         2
STATUS OF ARGLIST:
        A
        B
SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.
STATUS OF ASTK:
STATUS OF CSTK:
STATUS OF ARGLIST:
        A
        B
```

```
SUBROUTINE PARSER ENTERING CASE 4.

SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.

STATUS OF ASTK:
         3

STATUS OF CSTK:
         3

STATUS OF ARGLIST:
         A
         B
         .
```

1	STRING
FSTRING	
3	2
1	2
2	LEN
FLEN	
3	0
4	1
3	BREAK
FBREAK	
0	0
3	1

```
THE STRING TO BE MATCHED IS:  'SNOBOL4 PROGRAM.'.
THE LENGTH OF INSTRING IS 16.
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - AB
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MLEN.
MLEN MATCHED SUBSTRING 'SNOB'.
THE VALUE OF NEEDLE -  3.
THE VALUE OF CURSOR -  5.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MBREAK.
MBREAK SKIPPED SUBSTRING 'O'.
MBREAK SKIPPED SUBSTRING 'L'.
MBREAK SKIPPED SUBSTRING '4'.
MBREAK SKIPPED SUBSTRING ' '.
MBREAK SKIPPED SUBSTRING 'P'.
MBREAK SKIPPED SUBSTRING 'R'.
MBREAK SKIPPED SUBSTRING 'O'.
MBREAK SKIPPED SUBSTRING 'G'.
```

```
MBREAK SKIPPED SUBSTRING 'R'.
MBREAK SKIPPED SUBSTRING 'A'.
MBREAK SKIPPED SUBSTRING 'M'.
MBREAK FOUND BREAK CHARACTER '.'.
FIRST -  1
LAST - 16
SUBROUTINE MATCH MATCHED SUBSTRING 'SNOBOL4 PROGRAM'.
PATTERN MATCHING SUCCEEDED.


THE STRING TO BE MATCHED IS:  'SNOWWHITE'.
THE LENGTH OF INSTRING IS  9.
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - AB
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MLEN.
MLEN MATCHED SUBSTRING 'SNOW'.
THE VALUE OF NEEDLE -  3.
THE VALUE OF CURSOR -  5.
STATUS OF CPOS:
STATUS OF ASTK:
```

```
SUBROUTINE MATCH ENTERING SUBROUTINE MBREAK.
MBREAK SKIPPED SUBSTRING 'W'.
MBREAK SKIPPED SUBSTRING 'H'.
MBREAK SKIPPED SUBSTRING 'I'.
MBREAK SKIPPED SUBSTRING 'T'.
MBREAK SKIPPED SUBSTRING 'E'.
BREAK CHARACTERS NOT FOUND.
PATTERN MATCHING FAILED.
```

THE PATTERN TO BE USED IS:

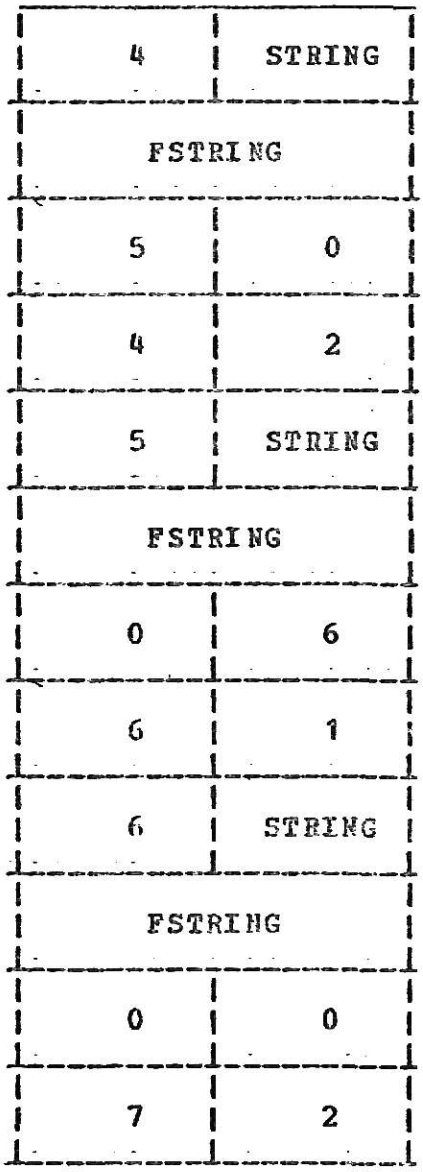
```
('B' | 'R') ('E' | 'EA') ('D' | 'DS')
SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.
SUBROUTINE PARSER ENTERING SUBROUTINE STRING.
SUBROUTINE PARSER ENTERING CASE 5.
STATUS OF ASTK:
       98

STATUS OF CSTK:
       98
        1

STATUS OF ARGLIST:
       B

SUBROUTINE PARSER ENTERING CASE 2.
SUBROUTINE PARSER ENTERING SUBROUTINE STRING.
STATUS OF ASTK:
       98
        2

STATUS OF CSTK:
       98
        1
        2

STATUS OF ARGLIST:
       B
       R

SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.
STATUS OF ASTK:

STATUS OF CSTK:

STATUS OF ARGLIST:
       B
       R
```

```
SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

SUBROUTINE PARSER ENTERING CASE 5.

STATUS OF ASTK:
        98

STATUS OF CSTK:
        98
         3

STATUS OF ARGLIST:
        B
        R
        E

SUBROUTINE PARSER ENTERING CASE 2.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

STATUS OF ASTK:
        98
         4

STATUS OF CSTK:
        98
         3
         4

STATUS OF ARGLIST:
        B
        R
        E
        E
        A

SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.

STATUS OF ASTK:

STATUS OF CSTK:

STATUS OF ARGLIST:
        B
        R
        E
        E
        A
```

```
SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

SUBROUTINE PARSER ENTERING CASE 5.

STATUS OF ASTK:
       98

STATUS OF CSTK:
       98
        5

STATUS OF ARGLIST:
       B
       R
       E
       E
       A
       D

SUBROUTINE PARSER ENTERING CASE 2.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

STATUS OF ASTK:
       98
        6

STATUS OF CSTK:
       98
        5
        6

STATUS OF ARGLIST:
       B
       R
       E
       E
       A
       D
       D
       S

SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.

STATUS OF ASTK:
       98
        6
       99
```

```
STATUS OF CSTK:
        98
         5
         6
        99

STATUS OF ARGLIST:
        B
        R
        E
        E
        A
        D
        D
        S
```

1	STRING
FSTRING	
3	2
1	1
2	STRING
FSTRING	
3	0
2	1
3	STRING
FSTRING	
5	4
3	1



THE STRING TO BE MATCHED IS: 'I READ.'.

THE LENGTH OF INSTRING IS 7.

THE VALUE OF NEEDLE - 1.

THE VALUE OF CURSOR - 1.

STATUS OF CPOS:

STATUS OF ASTK:

```
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - B
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - R
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  2.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - B
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  2.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - R
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  3.
STATUS OF CPOS:
STATUS OF ASTK:
```

```
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - B
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  3.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - R
MSTRING MATCHED SUBSTRING 'R'.
THE VALUE OF NEEDLE -  3.
THE VALUE OF CURSOR -  4.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - E
MSTRING MATCHED SUBSTRING 'E'.
THE VALUE OF NEEDLE -  5.
THE VALUE OF CURSOR -  5.
STATUS OF CPOS:
       1
STATUS OF ASTK:
       4
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - D
THE VALUE OF NEEDLE -  6.
THE VALUE OF CURSOR -  5.
```

```
STATUS OF CPOS:
        1

STATUS OF ASTK:
        4

SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.

STR - DS

THE VALUE OF NEEDLE -  4.

THE VALUE OF CURSOR -  4.

STATUS OF CPOS:

STATUS OF ASTK:

SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.

STR - EA

MSTRING MATCHED SUBSTRING 'EA'.

THE VALUE OF NEEDLE -  5.

THE VALUE OF CURSOR -  6.

STATUS OF CPOS:

STATUS OF ASTK:

SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.

STR - D

MSTRING MATCHED SUBSTRING 'D'.

FIRST -  3

LAST -  7

SUBROUTINE MATCH MATCHED SUBSTRING 'READ'.

PATTERN MATCHING SUCCEEDED.
```

THE PATTERN TO BE USED IS:

```
('XY' ('A' | 'B') | 'C')  'D'
SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.
SUBROUTINE PARSER ENTERING SUBROUTINE STRING.
SUBROUTINE PARSER ENTERING CASE 5.
STATUS OF ASTK:
        98
         1

STATUS OF CSTK:
        98

STATUS OF ARGLIST:
        X
        Y

SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.
SUBROUTINE PARSER ENTERING SUBROUTINE STRING.
SUBROUTINE PARSER ENTERING CASE 5.
STATUS OF ASTK:
        98
         1
        98

STATUS OF CSTK:
        98
        98
         2

STATUS OF ARGLIST:
        X
        Y
        A

SUBROUTINE PARSER ENTERING CASE 2.
SUBROUTINE PARSER ENTERING SUBROUTINE STRING.
STATUS OF ASTK:
        98
         1
        98
         3
```

```
STATUS OF CSTK:
        98
        98
         2
         3

STATUS OF ARGLIST:
        X
        Y
        A
        B

SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.

STATUS OF ASTK:
        98
         1

STATUS OF CSTK:
        98
        98
         2
         3
        99

STATUS OF ARGLIST:
        X
        Y
        A
        B

SUBROUTINE PARSER ENTERING CASE 2.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

STATUS OF ASTK:
        98
         1
         4

STATUS OF CSTK:
        98
        98
         2
         3
        99
         4
```

```
STATUS OF ARGLIST:
        X
        Y
        A
        B
        C

SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.

STATUS OF ASTK:

STATUS OF CSTK:

STATUS OF ARGLIST:
        X
        Y
        A
        B
        C

SUBROUTINE PARSER ENTERING CASE 2.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

STATUS OF ASTK:
         5

STATUS OF CSTK:
         5

STATUS OF ARGLIST:
        X
        Y
        A
        B
        C
        D
```

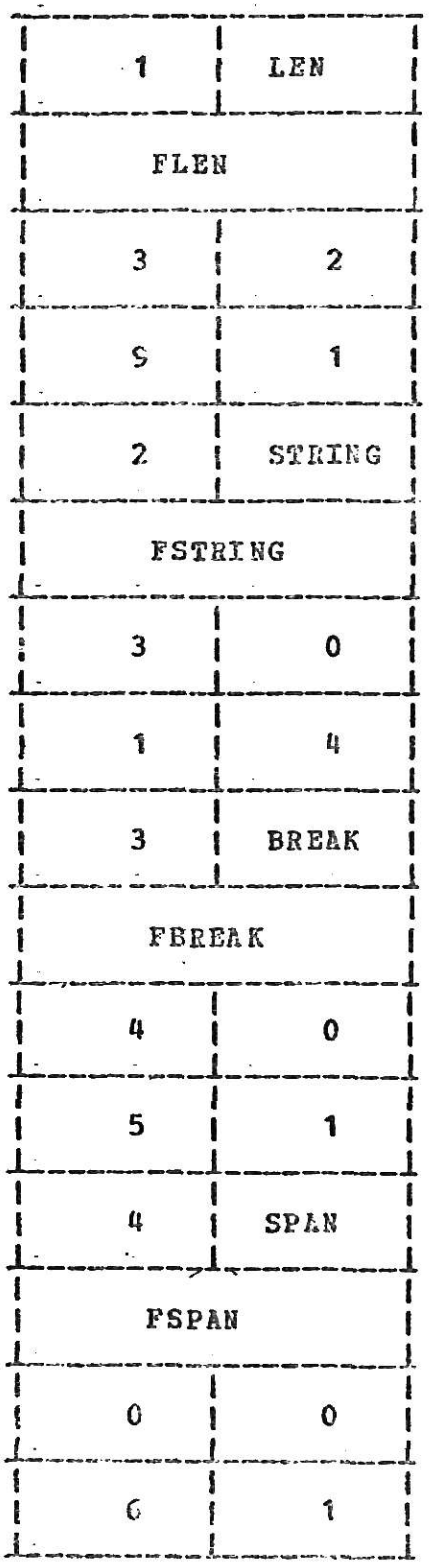
1	STRING
FSTRING	
2	4
1	2
2	STRING
FSTRING	
5	3
3	1
3	STRING
FSTRING	
5	0
4	1
4	STRING
FSTRING	
5	0
5	1
5	STRING
FSTRING	
0	0
6	1

```
THE STRING TO BE MATCHED IS:  'ABCDE'.
THE LENGTH OF INSTRING IS  5.
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - XY
THE VALUE OF NEEDLE -  4.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - C
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  2.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - XY
THE VALUE OF NEEDLE -  4.
THE VALUE OF CURSOR -  2.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - C
```

```
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  3.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - XY
THE VALUE OF NEEDLE -  4.
THE VALUE OF CURSOR -  3.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - C
MSTRING MATCHED SUBSTRING 'C'.
THE VALUE OF NEEDLE -  5.
THE VALUE OF CURSOR -  4.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - D
MSTRING MATCHED SUBSTRING 'D'.
FIRST -  3
LAST -  5
SUBROUTINE MATCH MATCHED SUBSTRING 'CD'.
PATTERN MATCHING SUCCEEDED.
```

```
THE PATTERN TO BE USED IS:

(LEN(9) | 'SNOB') BREAK('4') SPAN('4')

SUBROUTINE PARSER ENTERING CASE 1 - ( ENCOUNTERED.

SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.

SUBROUTINE PARSER ENTERING CASE 5.

STATUS OF ASTK:
        98

STATUS OF CSTK:
        98
         1

STATUS OF ARGLIST:

SUBROUTINE PARSER ENTERING CASE 2.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

STATUS OF ASTK:
        98
         2

STATUS OF CSTK:
        98
         1
         2

STATUS OF ARGLIST:
        S
        N
        O
        B

SUBROUTINE PARSER ENTERING CASE 3 - ) ENCOUNTERED.

STATUS OF ASTK:

STATUS OF CSTK:

STATUS OF ARGLIST:
        S
        N
        O
        B
```

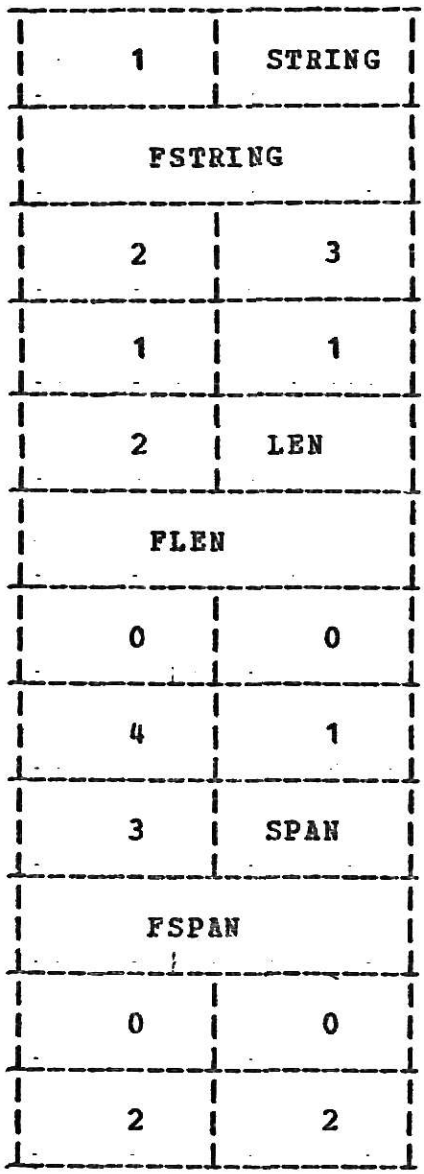
```
SUBROUTINE PARSER ENTERING CASE 4.

SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.

SUBROUTINE PARSER ENTERING CASE 5.

STATUS OF ASTK:
         2


STATUS OF CSTK:

STATUS OF ARGLIST:
         S
         N
         O
         B
         4

SUBROUTINE PARSER ENTERING CASE 4.

SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.

STATUS OF ASTK:
         3
         4

STATUS OF CSTK:
         4

STATUS OF ARGLIST:
         S
         N
         O
         B
         4
         4
```


1
LEN
FLEN
3
2
S
1
2
STRING
FSTRING
3
0
1
4
3
BREAK
FBREAK
4
0
5
1
4
SPAN
FSPAN
0
0
6
1

```
THE STRING TO BE MATCHED IS:  'SNOBOL4 PROGRAM'.

THE LENGTH OF INSTRING IS 15.

THE VALUE OF NEEDLE -  1.

THE VALUE OF CURSOR -  1.

STATUS OF CPOS:

STATUS OF ASTK:

SUBROUTINE MATCH ENTERING SUBROUTINE MLEN.

MLEN MATCHED SUBSTRING 'SNOBOL4 P'.

THE VALUE OF NEEDLE -  3.

THE VALUE OF CURSOR - 10.

STATUS OF CPOS:
        9

STATUS OF ASTK:
        2

SUBROUTINE MATCH ENTERING SUBROUTINE MBREAK.

MBREAK SKIPPED SUBSTRING 'R'.

MBREAK SKIPPED SUBSTRING 'O'.

MBREAK SKIPPED SUBSTRING 'G'.

MBREAK SKIPPED SUBSTRING 'R'.

MBREAK SKIPPED SUBSTRING 'A'.

MBREAK SKIPPED SUBSTRING 'M'.

BREAK CHARACTERS NOT FOUND.
```

```
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - SNOB
MSTRING MATCHED SUBSTRING 'SNOB'.
THE VALUE OF NEEDLE -  3.
THE VALUE OF CURSOR -  5.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MBREAK.
MBREAK SKIPPED SUBSTRING 'O'.
MBREAK SKIPPED SUBSTRING 'L'.
MBREAK FOUND BREAK CHARACTER '4'.
THE VALUE OF NEEDLE -  4.
THE VALUE OF CURSOR -  7.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSPAN.
MSPAN MATCHED SUBSTRING 4.
FIRST -  1
LAST -  8
SUBROUTINE MATCH MATCHED SUBSTRING 'SNOBOL4'.
PATTERN MATCHING SUCCEEDED.
```

```
THE PATTERN TO BE USED IS:

'A' LEN(4) | SPAN('XY')

SUBROUTINE PARSER ENTERING CASE 2.

SUBROUTINE PARSER ENTERING SUBROUTINE STRING.

SUBROUTINE PARSER ENTERING CASE 5.

STATUS OF ASTK:
         1

STATUS OF CSTK:

STATUS OF ARGLIST:
        A

SUBROUTINE PARSER ENTERING CASE 4.

SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.

SUBROUTINE PARSER ENTERING CASE 5.

STATUS OF ASTK:

STATUS OF CSTK:
         2

STATUS OF ARGLIST:
        A

SUBROUTINE PARSER ENTERING CASE 4.

SUBROUTINE PARSER ENTERING SUBROUTINE PATREC.

STATUS OF ASTK:
         3

STATUS OF CSTK:
         2
         3

STATUS OF ARGLIST:
        A
        X
        Y
```



THE STRING TO BE MATCHED IS: 'XYXYXYZ'.

THE LENGTH OF INSTRING IS 7.

THE VALUE OF NEEDLE - 1.

THE VALUE OF CURSOR - 1.

STATUS OF CPOS:

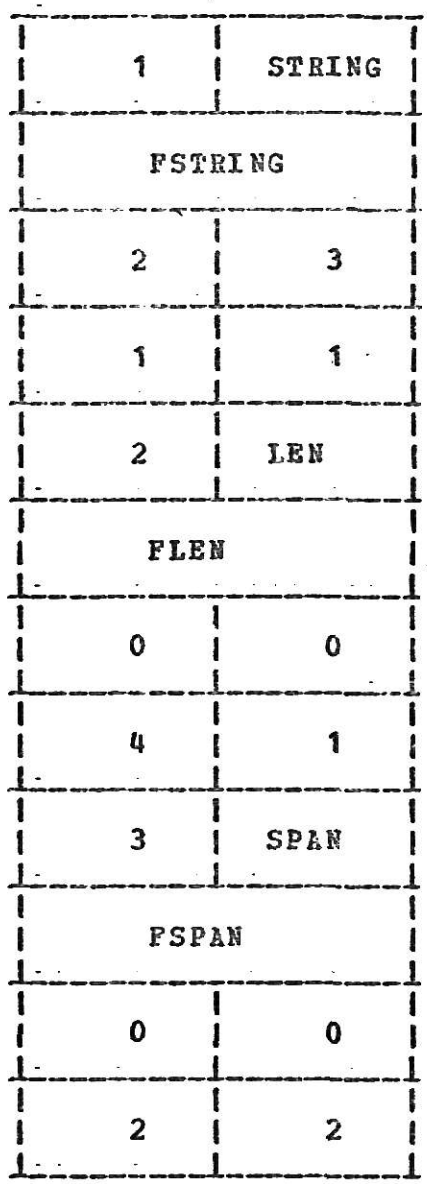
STATUS OF ASTK:

```
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - A
THE VALUE OF NEEDLE -  3.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSPAN.
MSPAN MATCHED SUBSTRING X.
MSPAN MATCHED SUBSTRING Y.
MSPAN MATCHED SUBSTRING X.
MSPAN MATCHED SUBSTRING Y.
MSPAN MATCHED SUBSTRING X.
MSPAN MATCHED SUBSTRING Y.
FIRST -  1
LAST -  7
SUBROUTINE MATCH MATCHED SUBSTRING 'XYXXXY'.
PATTERN MATCHING SUCCEEDED.


THE STRING TO BE MATCHED IS:  'FANTASTIC'.
THE LENGTH OF INSTRING IS  9.
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
```

```
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - A
THE VALUE OF NEEDLE -  3.
THE VALUE OF CURSOR -  1.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSPAN.
THE VALUE OF NEEDLE -  1.
THE VALUE OF CURSOR -  2.
STATUS OF CPOS:
STATUS OF ASTK:
SUBROUTINE MATCH ENTERING SUBROUTINE MSTRING.
STR - A
MSTRING MATCHED SUBSTRING 'A'.
THE VALUE OF NEEDLE -  2.
THE VALUE OF CURSOR -  3.
STATUS OF CPOS:
        1
STATUS OF ASTK:
        3
SUBROUTINE MATCH ENTERING SUBROUTINE MLEN.
MLEN MATCHED SUBSTRING 'NTAS'.
FIRST -  2
LAST -  7
SUBROUTINE MATCH MATCHED SUBSTRING 'ANTAS'.
PATTERN MATCHING SUCCEEDED.
```

THE PATTERN TO BE USED IS:

'A' LEN(4) | SPAN('XY')



THE STRING TO BE MATCHED IS: 'XYXYXYZ'.

FIRST - 1

LAST - 7

SUBROUTINE MATCH MATCHED SUBSTRING 'XYXYXY'.

PATTERN MATCHING SUCCEEDED.

THE PATTERN TO BE USED IS:

'A' | 'B' 'C' | 'D'

1	STRING
FSTRING	
0	2
1	1
2	STRING
FSTRING	
3	4
2	1
3	STRING
FSTRING	
0	0
3	1
4	STRING
FSTRING	
0	0
4	1

```
THE STRING TO BE MATCHED IS:  'BCDE'.

FIRST -  1

LAST -  3

SUBROUTINE MATCH MATCHED SUBSTRING 'BC'.

PATTERN MATCHING SUCCEEDED.


THE STRING TO BE MATCHED IS:  'ABCD'.

FIRST -  1

LAST -  2

SUBROUTINE MATCH MATCHED SUBSTRING 'A'.

PATTERN MATCHING SUCCEEDED.
```

THE PATTERN TO BE USED IS:

('A' | 'B') ('C' | 'D')

1	STRING
FSTRING	
3	2
1	1
2	STRING
FSTRING	
3	0
2	1
3	STRING
FSTRING	
0	4
3	1
4	STRING
FSTRING	
0	0
4	1

```
THE STRING TO BE MATCHED IS:  'BCDE'.

FIRST -  1

LAST -  3

SUBROUTINE MATCH MATCHED SUBSTRING 'BC'.

PATTERN MATCHING SUCCEEDED.
```

IMPLEMENTATION OF SNOBOL4 PATTERN MATCHING

by

JOSEPHINE LI-MING LIU

B.A., Fu Jen Catholic University

Taipei, Taiwan, ROC 1969

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1977

ABSTRACT

The goal of this project was to design and implement a program that performs parsing and pattern matching for a subset of SNOBOL4 patterns. The program reads in a SNOBOL4 pattern, evaluates it and creates a pattern structure. Then using the pattern structure, it performs pattern matching on a subject string. When an error occurs, such as an illegal pattern detected during parsing, the program will print out an error message before continuing with the next parsing and pattern matching task. Otherwise, at the end of each parsing and pattern matching task, the program will print out a message indicating whether or not pattern matching succeeded and, if it succeeded, the substring matched.

Before the design phase of the project, the SNOBOL4 pattern matching operation was studied and several simple SNOBOL4 programs were run and examined for their output. Next the basic data structures and the major modules of the program were designed. Structured programming techniques were used in designing and coding the program which consists of a driver routine and nine subroutines, totally about six hundred and thirty lines of source code. A tracing facility was included for easier debugging.

Several different sets of test data were created to test the correctness of the program. The program has been tested extensively and the results were successful for each set of test data.