AD-A048 023 WEAPONS RESEARCH ESTABLISHMENT SALISBURY (AUSTRALIA) F/G 9/2
A FINITE-STATE PARSER FOR PDP-11 AND NOVA COMPUTERS.(U)
JUL 77 G S BRIMBLE
UNCLASSIFIED WRE-TR-1842(A) NL

1 OF 1
ADA048023

END
DATE
FILMED
1 - 78
DDC

WRE-TR-1842 (A)

AR-000-600

AD A 048023

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

WEAPONS RESEARCH ESTABLISHMENT

SALISBURY, SOUTH AUSTRALIA

TECHNICAL REPORT 1842 (A)

A FINITE-STATE PARSER FOR PDP-11 AND NOVA COMPUTERS

G.S. BRIMBLE

Approved for Public Release

COPY No. 19

JULY 1977

UNCLASSIFIED

AR-000-600

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

WEAPONS RESEARCH ESTABLISHMENT

TECHNICAL REPORT 1842(A)

A FINITE-STATE PARSER FOR PDP-11 AND NOVA COMPUTERS

G.S. Brimble

SUMMARY

A method is described for creating computer programs to analyse text for compliance with a specified structure; examples of such programs are command decoders and format checkers. The paper shows how analysis programs are easily produced by constructing a table directly from a specification of the intended data format and presenting this as input to a non-specific parser subroutine which has been written and is described. Facilities are included to execute user-written subroutines when key structures are recognised in the input.

Programs have been written in PDP-11 and NOVA assembly language to implement the parser, and instructions for use of these programs, examples and listings are included.

Approved for public release.

POSTAL ADDRESS: The Director, Weapons Research Establishment, Box 2151, G.P.O., Adelaide, South Australia, 5001

UNCLASSIFIED

DOCUMENT CONTROL DATA SHEET

Security classification of this page: UNCLASSIFIED

1 DOCUMENT NUMBERS

AR Number:	AR-000-600
Report Number:	WRE-TR-1842(A)
Other Numbers:	

2 SECURITY CLASSIFICATION

a. Complete Document:	Unclassified
b. Title in Isolation:	Unclassified
c. Summary in Isolation:	Unclassified

3 TITLE

A FINITE-STATE PARSER FOR PDP-11 AND NOVA COMPUTERS

4 PERSONAL AUTHOR(S):

G.S. Brimble

5 DOCUMENT DATE:

July 1977

6
6.1 TOTAL NUMBER OF PAGES 38

6.2 NUMBER OF REFERENCES: 4

7
7.1 CORPORATE AUTHOR(S):

Weapons Research Establishment

7.2 DOCUMENT (WING) SERIES AND NUMBER

Applied Physics Wing
TR-1842

8 REFERENCE NUMBERS

a. Task: DST 76/190

b. Sponsoring Agency:

9 COST CODE:

185256

10 IMPRINT (Publishing establishment):

Weapons Research Establishment

11 COMPUTER PROGRAM(S)
(Title(s) and language(s))

TESTP - MACRO 11
PARSE - MACRO 11
TESTP - NOVA ASS.L.
PARSE - NOVA ASS.L.

12 RELEASE LIMITATIONS (of the document):

Approved for public release

12.0	OVERSEAS	NO		P.R.	1	A		B		C		D		E	

Security classification of this page: UNCLASSIFIED

3

Security classification of this page: UNCLASSIFIED

13 ANNOUNCEMENT LIMITATIONS (of the information on these pages):

No limitation

14 DESCRIPTORS:

a. EJC Thesaurus Terms

computer programs
analyzing
subroutines

b. Non-Thesaurus Terms

text analysis
command decoders
format checkers
parsers
PDP-11
NOVA

15 COSATI CODES:

0902

16 LIBRARY LOCATION CODES (for libraries listed in the distribution):

SW SD SR AACA

17 SUMMARY OR ABSTRACT:
(if this is security classified, the announcement of this report will be similarly classified)

A method is described for creating computer programs to analyse text for compliance with a specified structure; examples of such programs are command decoders and format checkers. The paper shows how analysis programs are easily produced by constructing a table directly from a specification of the intended data format and presenting this as input to a non-specific parser subroutine which has been written and is described. Facilities are included to execute user-written subroutines when key structures are recognized in the input.

Programs have been written in PDP-11 and NOVA assembly language to implement the parser, and instructions for use of these programs, examples and listings are included.

Security classification of this page: UNCLASSIFIED

TABLE OF CONTENTS

LIST OF APPENDICES

1. INTRODUCTION

A commonly occurring requirement in assembly language programming is analysis of input text strings to implement activities described by that input. Examples are typed commands to a real-time program or mnemonic-encoded instructions to an assembler. Programmers tend to solve this problem by writing a decoder routine specifically tailored to the expected format; if it is permissible in the application, the format is usually compressed to be as cryptic as possible to simplify this task.

Another solution is to use a general-purpose decoder routine which operates on a coded description of the format expected in the data to detect the presence of salient structures and direct implementation of appropriate actions. This method allows the programmer to concentrate on the description of the data format and the execution of the actions invoked by the input, and results in easier writing and debugging. In addition, a rich and natural command format is encouraged and expansion is easy to accommodate.

The code produced by using a table-driven parser is often very similar in execution to an ad hoc decoder, while being far easier to understand. The features included in the parser can be tailored to suit the size and complexity of the job; whilst some overhead is inevitable, for large or complex formats the efficiency of the code produced by the two techniques is comparable.

The routines described here were originally developed to decode keyboard commands on a real-time data logger and have since been used for text format verification. They are principally designed to recognize structures encountered in command strings, but other structures are easily included.

2. THEORY

2.1 Preliminaries

A good way to describe an artificial language of the kind commonly used in computing is by a phrase-structure grammar(ref.1). This technique consists of grouping together basic structural elements to form more complex items, which in turn are grouped to build up the language. An example is a natural language like English, where letters are grouped to form words, words to make phrases, clauses etc. and these parts to make sentences. The grammar is the set of rules which defines how such combinations are made, i.e. the rules describe the syntax of the language.

It is also necessary to use a language to describe the grammar; some way must be found to set out the rules of syntax. This can be done in English, but problems of inexactness of meaning arise with such a rich language. A simpler and more formal language (a meta-language) is preferable for such description. Backus-Naur Form (BNF) is a meta-language which was developed to describe syntactic rules unambiguously. A form of BNF will be used here; a summary of the elements of BNF is included in Appendix I(ref.2).

2.2 Table-driven parsers

The process of examining a data sample to determine whether it conforms with a particular grammar is parsing. A simple strategy for designing a parser program is to develop a table which directs a set of general routines to carry out the parse; the table is constructed from the syntax rules of the grammar being analysed. This is a table-driven parser; the table specifies the transitions or changes which occur between states or elements within the grammar. The table is a transition table or state table(ref.3).

The parsing procedure consists of starting at the top or head state of the transition table and comparing the first element of the input with the syntactic types which identify the columns in the table. Where a match occurs, a transition is specified by a pointer in the table to the next state or row. This process continues until a transition is found to a special state indicating success or failure. This is best seen by example.

2.3 Constructing a transition table

A language is described below in BNF(ref.3). It consists of simple arithmetic assignment statements. The elements which make up the language are the meta-components <letter>, <null>, =, and <operator> as defined from basic objects in Appendix I.

<assignment statement> ::= <letter> <rest of a.s.>

<rest of a.s.> ::= <expression>

<expression> ::= <letter> <rest of exp.>

<rest of exp.> ::= <null> | <operator> <expression>

This grammar allows such constructs as A = B + C and D = A * G - D. The transition table constructed directly from this grammar is:

State	Meta component			
	<letter>	<operator>	=	<null>
1. <assignment statement>	2	5	5	5
2. <rest of a.s.>	5	5	3	5
3. <expression>	4	5	5	5
4. <rest of exp.>	5	3	5	6

5. ERROR
6. SUCCESS

The table is built by considering each definition in turn. If the first meta-component on the right-hand side is found, the second must be found next so it becomes the target for the next stage of the parse; thus a pointer is entered in the table, specifying a transition to that state. The parse process for the statement A = B proceeds as follows:

(a) Entering at state 1, the first input element A matches <letter> and a transition occurs to state 2. Anything which did not start with a letter, e.g. the statement - B = C, would be another object and fail the parse.

(b) In state 2, the = causes a transition to state 3; statements like A + B = C would fail.

(c) B matches <letter> in state 3 and a transition occurs to state 4 where <null> causes success. Here A = BC would fail, but A = B + C would proceed because the + would match an <operator> and a transition to state 3 would occur.

2.4 Language restriction

The above language definition could be simplified if the first state was changed to:

<assignment statement> ::= <letter> = <expression>

However the transition table cannot handle this structure. The intermediate state <rest of a.s.> was necessary so that in the right-hand side of the definition, a single object is followed by a single meta-component. This ensures that at any stage in the parse no backtracking is necessary if the route being followed fails(ref.3). Parsing is then fast and efficient at the cost of restricting the range of grammars which can be handled. Application of this rule will produce a right linear regular grammar, i.e. a finite state (Chomsky type 3) grammar(ref.1). Simple precedence languages are produced by finite state grammars(ref.3) and these ensure a unique parse tree through the grammar for any input. Every finite language can be generated by a regular grammar(ref.1), and the majority of languages used in computing are finite (i.e. consist of a finite number of combination of elements).

A program to analyse according to a regular grammar is a finite-state parser.

2.5 Subsidiary transition tables

A situation can occur where a particular meta-component constructed in the grammar could be used as a pseudo-object within the grammar. An example of this occurs if the example of Section 2.3 is modified to require three letter variable names. Then the BNF definition becomes:

<assignment statement> ::= <letter> <rest of a.s.1>

<rest of a.s.1> ::= <letter> <rest of a.s.2>

<rest of a.s.2> ::= <letter> <rest of a.s>

<rest of a.s.> ::= <expression>

<expression> ::= <letter> <rest of exp. 1>

<rest of exp. 1> ::= <letter> <rest of exp. 2>

<rest of exp. 2> ::= <letter> <rest of exp.>

<rest of exp.> ::= <null> | <operator> <expression>

The same kind of structure has been invoked each time the variable name appears. It would be more efficient to define a pseudo-object in a subsidiary transition table which can then be used in the main table as if it were an object. A successful parse in the subsidiary table would be equivalent to matching the object. The effect of the subexpression structure is similar to the subroutine concept in programming. The BNF definition and transition tables for the example would consist of a main table and definition as in Section 2.3, but with <letter> replaced by <name>, and the subsidiary structures:

<name> ::= <letter> <more name>

<more name> ::= <letter> <rest of name>

<rest of name> ::= <letter>

	<letter>	=	<operator>	<null>
1. <name>	2	5	5	5
2. <more name>	3	5	5	5
3. <rest of name>	4	5	5	5
4. RETURN				
5. ERROR				

In this particular example another solution could have been found without using the subexpression structure. If the definition of a variable name could be changed to be one or more letters unrestricted in length, then powerful recursive definitions can be used thus:

<assignment statement> ::= <letter> <rest of a.s.>

<rest of a.s.> ::= <letter> <rest of a.s> | = <expression>

<expression> ::= <letter> <rest of exp.>

<rest of exp.> ::= <letter> <rest of exp.> | <null> | <operator> <expression>

2.6 Semantics

The parser merely determines whether or not the input is syntactically correct. Its only output is logical - the sentence parsed or it did not. Normally, the input is intended to trigger some operation within the program depending on its content; that is, the input has meaning or semantics in the context of the program, and it is examined in order to decode the semantics. Semantic structure is much more difficult to formalise and describe. However, in a practical table-driven parser, it is possible to allow execution of subroutines each time a transition occurs. This permits the programmer to execute code in his program at key points during the parse when particular structures are recognized, and this is generally adequate for implementing semantics.

3. IMPLEMENTATION

3.1 Discussion

In order to implement a finite-state parser on a computer, it is necessary to have a set of routines each of which recognizes one object of the language, a formalism for storing the transition table and an executive to direct the matching and transition process. In practice, some extensions permit greater efficiency(ref.4).

3.2 Transition table storage

The transition table will be stored in a computer as a list because of the sequential nature of computer memory. This is easily accomplished and a number of redundant transitions removed, by altering the table construction. A list can be made consisting of a series of cells, one for each transition. The first entry of each cell names the meta-component or object to be matched and is equivalent to the column heading in the table; the next entry points to the target state for the transition. Furthermore, all meta-components which are not allowed in a particular state, i.e. will produce an error, can be designated by the pseudo-object <anything else>. Then the transition table for the example of Section 2.3 is as shown in figure 1(a).

Normally symbolic labels are used in assembly language, and thus as state pointers. The object <anything else> always points to the ERROR exit so this cell can be contracted to a single "end of state" code which is recognized by the parser executive. The final list for the example is smaller than the original table (figure 1(b)). When many objects exist in the language the improvement is substantial.

3.3 Pseudo-objects

In most computing applications, structures like words and numbers are needed as basic elements along with individual ASCII characters like key letters or punctuation marks. Conceptually these applications can be satisfied by defining the entire character set as objects and constructing words and numbers in subsidiary tables as explained in Section 2.5. However, it is more efficient to include words and numbers as pseudo-objects by providing routines in the parser to recognize them directly. The objects which this parser recognizes include the individual characters of the ASCII set, any arbitrary character, specific words of any length, any arbitrary letter string, integer numbers, and end-of-line marks. Other objects can be added by users who write the appropriate recognition routines.

3.4 Action routines

A facility to execute user-written subroutines is provided by expanding the cell in the transition table to include a pointer to a subroutine which is executed if the transition is taken. The parser includes a null subroutine which does nothing; then in those transitions where no action is needed a pointer to this subroutine is placed. Then the cell of the transition table list is:

```
                 .
                 .
                 .
(state label:)  code indicating type of object to match
                (any trailing arguments)
                pointer to action subroutine if match succeeds
                pointer to target state if match succeeds
                 .
                 .
                 .
                end of state code
                 .
                 .
                 .
```

3.5 Defaults

A useful feature arises if another pseudo-object is defined which always succeeds so that its action routine and transition are always implemented. Its usefulness lies in its ability to allow defaults in the input string. For example, a command structure might be decoded to determine user-entered parameters; if a particular parameter is missing, the test for it would fail but by following this with the matching object the parse can proceed to the next parameter, leaving an initialised default value. A specific example of this is set out in Appendix II.

3.6 Construction

The parser is written as a subroutine which is called from the user's main program. The objects or syntactic types which are recognized are defined as numeric codes for constructing the transition table. Pointers to this table and the text string to be analysed are passed in the subroutine call. Analysis proceeds in the parser by using the object codes to vector to a routine to match that object; if the match occurs a success return executes the action subroutine for the transition, then selects the target state to find the next object code. If the match fails, a failure return skips the action routine and target pointers to select the object from the following transition. If the end-of-state code is found, control returns to the calling program with an error code; if a success state is entered, a success code is returned. If the parse failed a pointer to the place in the input where failure occurred is available.

4. HOW TO USE THE PARSER

4.1 Calling sequence for PDP-11

The subroutine call 'JSR PC, PARSE' initiates parsing. Register RØ will contain the result of the parse on return - RØ = Ø if successful, = -2 if failed. Registers R3 to R6 are unaffected by the parse, although one word of stack space is temporarily used for each level of subexpression evaluation. The parser returns to the caller by 'RTS PC'.

When the call occurs, R1 must point to the first byte in a byte array of 7-bit ASCII text to be analysed. If the parse fails, R1 will point to the first unrecognized byte. Register R2 must point to the head state of the transition table defining the grammar which the text must obey. The cells of this table have the format

syntactic type code

any trailing arguments

address of action subroutine (pointer for indirect jump)

address of target state

The end-of state code is -1. The address of the "do nothing" subroutine is available by using the symbol "NULL." The parser executes action subroutines by 'JSR PC, @ POINTER' so they must resume the parse by 'RTS PC'. Action subroutines must preserve registers R1 and R2 and tidy the stack if used.

The parser is assembled into its own program section to avoid symbol clash. The symbols 'NULL.' and 'PARSE' are declared global, as are 'NUM.' and 'CHAR.' (see Section 4.3).

4.2 Calling sequence for NOVA

The instruction 'JSR PARSE' will initiate parsing. This passes the return address to the parser in AC3. Accumulator AC1 must hold a word pointer to the text string to be analysed, stored as 7 bit ASCII bytes in .TXTM 1, i.e. the first character in the top byte (MSB's) then the second in the low byte. AC2 must contain a pointer to the head state of the transition table which has the format for each cell

syntactic type code

any trailing arguments

address of action subroutine

address of target state

The end-of-state code is -1. The user must define page zero locations for CHAR. and NUM. and the mask variable IDMSK containing 377 octal. The parser uses locations 20, 21 and 30. Auto increment register 20 is assigned the label STPT by the parser and is used to hold the pointer to the state table; 21 is TXPT and points to the text string word by word, and 30 is STACK and indexes a 10-word stack which allows 5 levels of nesting of subexpressions.

After parsing, ACØ carries back a result code to the caller. If a successful parse occurred, ACØ = Ø; if it failed, ACØ = -1. If the parse fails, the page zero register 21 (TXPT) is pointing to the word containing the byte which caused the failure, except that if the first byte in the string fails TXPT will point one word before it because of auto increment management.

Action routines are called as subroutines via 'JSR @ POINTER' so they must resume the parse by 'JMP Ø, 3' or similar arrangement. No other accumulators need preservation by routines. The 'do-nothing' subroutine provided by the parser is at address 'NULL.'.

4.3 Syntactic types

The following entries indicate the octal number code for each syntactic type with the symbols assigned to them by the parser in definition statements, plus a brief description of the objects they represent.

4.3.1 ALPH. = 1

This matches all following alphabetic characters until a non-alphabetic character is found. It amounts to an "ignore text" specification, allowing optional extensions on key symbols, etc. The entry has a trailing one-word argument. The low byte is a maximum byte count to limit the amount of text to be ignored, with Ø corresponding to the largest limit (256 characters). The high byte is a flag indicating whether spaces and tabs count as alphabetic or non-alphabetic characters; flag = Ø if the former, 1 if the latter. If flag = 1, the match will cease if a space or tab is found. This type only fails if the character count is exceeded. Note that it succeeds if there are no alphabetic characters, since its effect is to push the text pointer on to the next non-alphabetic character, which can be the next byte.

4.3.2 NUMB. = 2

This matches a decimal integer whose value is stored in the parser-defined location 'NUM.' (global) in a PDP-11 or 'NUM.' (page zero) in a NOVA. Conversion stops at the first non-numeric character. It always succeeds - if no numerals are present, NUM. = Ø.

4.3.3 KEY. = 3

Key words are one or more specific ASCII characters, e.g. "NAME" or "*". These follow the type specification in the transition table as trailing arguments, and terminate with a null byte. The NOVA assembler will correctly store them if in .TXTM 1 mode the .TXT pseudo-operator is used. The PDP-11 MACRO assembler provides a .ASCIZ pseudo-operator; the .EVEN pseudo-operator must be used after the string to return to word boundaries. For single character keywords, a literal argument is possible - for PDP-11 use '.BYTE 'X,Ø' and for NOVA '256.*"X'.

4.3.4 EOF. = 4

Equivalent to the null operator in the examples of Section 2.3, this matches ESC, ALT MODE, FORM FEED, RUBOUT, CR codes. On a PDP-11, if a CR is found and it is trailed by a LF, the LF is also matched and the text pointer will indicate the character after the LF. The particular terminator found is available to the action subroutine in register R3. In a NOVA the particular character is in ACØ.

4.3.5 MTCH. = 5

This always succeeds. It does not move the text pointer. It forces action and/or a new state.

4.3.6 ANY. = 6

This puts the next character into the parser-defined location 'CHAR.' (global) on a PDP-11 or 'CHAR.' (page zero) on a NOVA. It succeeds for all characters except EOF (14) or rubout (377).

4.3.7 SUBX. = 7

This code asks the parser to match a subexpression, i.e. to attempt to match a structure defined by a subsidiary transition table. The type entry is followed by a single word trailing argument containing the address of the head state of the subsidiary table. This subexpression state table must contain a state with the type 'ENDX.' as the success exit to return to the main table.

4.3.8 ENDX. = 1Ø

This is the success exit from a subexpression. It causes the parser to carry out the success action and target in the transition which called the subexpression. It has no action or target entries of its own in the subsidiary transition table.

4.3.9 END. = Ø

This type indicates successful parse and must be present in the transition table. It cannot be included in a subsidiary transition table for a subexpression. It causes the parser to return control to the user's program with the success code in RØ or ACØ. It is the only exit for a successful parse. It has no action or target entries associated with it.

5. CONCLUSION

It has been found in practice that programs can be written and debugged more easily and faster using the structure described, compared with individual algorithms for text decoding. The routines described and listed in the appendices have been used and found to match their descriptions. It is recommended that consideration should be given to using them in any text or command handling routines, or to implementing a similar structure for special purposes. Their use can save considerable program development time.

REFERENCES

No.	Author	Title
1	Cardenas, A.F. et al	"Computer Science". Wiley (N.Y.), 1972.
2	Naur, P. (ed.)	"Revised Report on the Algorithmic Language ALGOL 68" Comm. ACM 6, January 1963
3	Maginnis, J.B.	"Elements of Compiler Construction" Appleton - Century - Crofts (N.Y.), 1972
4	-	"TPARS Users Manual". RSX-11D V6A System Release Notes.

APPENDIX I

BACKUS-NAUR FORM (BACKUS NORMAL FORM)

BNF is a formalism for describing languages which consists of a set of definition statements. It contains two metalinguistic connectives which cannot be part of the language being described. These are

::= meaning "is defined by"; it is a single symbol

| the logical OR operator

The other components of the formalism are metalinguistic variables, being sequences of characters enclosed in angle brackets < > having symbolic meaning, and objects or marks which are basic undefinable elements of the language being described. Juxtaposition of marks and/or variables in a formula signifies juxtaposition of the sequences denoted. Statements in BNF consist of concatenation of metalinguistic variables, metalinguistic connectives and objects in the form

<metacomponent> ::= <variable> <variable> .....

For example, the statement

<operator> ::= +

is a valid BNF definition. The variable <operator> is defined as the symbol + which is an object ("plus sign" - it cannot itself be defined).
Similarly

<operator> ::= -

<operator> ::= *

<operator> ::= /

define alternative meanings of the variable or meta-component <operator>. This definition is made more convenient by using the OR operator

<operator> ::= +| -| *| /

Similarly

<digit> ::= 1| 2| 3| 4| 5| 6| 7| 8| 9| Ø

Then more powerful meta-components are built up by combining objects and meta-components thus:

<integer> ::= <digit> | <digit> <integer>

<sign> ::= +| -

<fractional part> ::= <digit> | <fractional part> <digit>

<unsigned number> ::= <integer> | <integer> .<fractional part>

<number> ::= <sign> <unsigned number> | <unsigned number>

The meta-components <null> and <empty> signify the null set of symbols. In a practical line-structured input they would correspond to end of line.

APPENDIX II

AN EXAMPLE OF PARSER USE

The same example is coded below for both PDP-11 and NOVA computers. The PDP-11 program uses RSX-11D operating system directives for input and output, and the NOVA program uses RTOS directives; however both should still be comprehensible without understanding these directives.

The example decodes a command string typical of an RSX system command (it requests creation of a User File Directory entry on a storage unit). The command has a mandatory format specifying a device and a user code

```
UFD_XX:[N,N]
```

where __ represents space, X represents letters and N digits.

A number of options can be included, and these are shown in brackets

```
UFD__XX(N):(XX.....)[N(NN.....), N(NN.....)](/PRO = [R,R,R,R])
                                 (/ALLOC = NN.....)
```

where R represents any combination of <null> | R| W| E| D

Thus a legitimate command incorporating all options would be

```
UFD__DK1:SCRATCH [200,200] /PRO = [RWED, RW,,R] /ALLOC = 300
```

Action routines are not shown in the examples but would be assembly language subprograms to cause the system executive to carry out the operations specified by the command.

The finite-state grammar, in BNF using pseudo-objects available in the parser, follows:

```
<command>        ::=  UFD__<next 1>
<next 1>         ::=  <letter> <next 2>
<next 2>         ::=  <letter> <next 3>
<next 3>         ::=  <number> <device term.> | <device term.>
<device term.>   ::= : <next 4>
<next 4>         ::=  <word> <next 5> | <next 5>
<next 5>         ::=  <uic> <opts>
<opts>           ::=  <null> | /<next 6>
<next 6>         ::=  ALLOC = <alc> | PRO = <pro>
<a|c>            ::=  <number> <opts>
<pro>            ::=  [<rest pro>
<rest pro>       ::=  ]<opts> | ,<rest pro> | R <rest pro> | W <rest pro> |
                           E <rest pro> | D <rest pro>
```

Here <uic> has been used as a pseudo-object for illustration. It must be defined in a subsidiary definition.

```
<uic>            ::=  [<u1>
<u1>             ::=  <number> <u2>
<u2>             ::=  ,<u3>
<u3>             ::=  <number> <u4>
<u4>             ::=  ]<null>
```

These definitions are translated directly into transition tables in the examples whose listings follow

APPENDIX III

PDP-11 PARSER PROGRAM LISTING

```
TESTP - TEST PAPSER      MACRO 01013  24-AUG-77 12:30  PAGE 1

                                     .TITLE TESTP - TEST PARSER
                                     ;
                                     ;G. CRIMBLE 30/9/76
                                     ;MODIFIED TO NEW TYPE SYMBOLS 26/1/77
                                     ;
                                     ; WILL LINK TO PARSER.OBJ AT TKB
                                     ;
        000000                               END.=0
        000001                               ALPH.=1
        000002                               NUMB.=2
        000003                               KEY.=3
        000004                               EOF.=4
        000005                               MTCH.=5
        000006                               ANY.=6
        000007                               SUBX.=7
        000010                               ENDX.=10
                                     ;
                                             .MCALL  EXIT$S  DIR$    QIOW$
                                     ;
000000                               TESTP:  DIR$    #PRMPT          ;INDICATE READY
000006  103774                               BCS     TESTP
000010                                       DIR$    #CML            ;GET COMMAND
000016  103770                               BCS     TESTP
000020  026727  000240  000000G              CMP     IOST,#IE.EOF    ;IS IT CTRL-Z?
000026  001441                               BEQ     EXIT            ;THEN FINISH
000030  016701  000232                       MOV     IOST+2,R1
000034  062701  000270'                      ADD     #CMB,R1         ;POINT TO LAST BYTE
000040  116711  000221                       MOVB    IOST+1,(R1)     ;STORE TERMINATING CHAR
000044  012701  000270'                      MOV     #CMB,R1         ;PREPARE TO PARSE
000050  012702  000412'                      MOV     #START,R2
000054  004767  000000G                      JSR     PC,PARSE        ;PARSE
000060  005700                               TST     R0              ;GOOD PARSE?
000062  001416                               BEQ     SUX             ;YES - INDICATE
000064  010167  000166                       MOV     R1,ERP+Q.IOPL   ;NO - PRINT WHY
000070  162701  000270'                      SUB     #CMB,R1         ;SET BYTE COUNT
000074  160167  000166                       SUB     R1,IOST+2
000100  016767  000162  000152               MOV     IOST+2,ERP+Q.IOPL+2
000106                                       DIR$    #ERP
000114  000167  177660                       JMP     TESTP
000120                               SUX:    DIR$    #SUXMES
000126  000167  177646                       JMP     TESTP
000132                               EXIT:   EXIT$S
000140                               PRMPT:  QIOW$   IO.WVB,5,24,,,,<PRPT,4,44>
000162     124     123     124       PRPT:   .ASCII'TST>'
000165     076
000166                               CML:    QIOW$   IO.RLB,5,24,,IOST,,<CMB,82.,40>
000210                               SUXMES: QIOW$   IO.WVB,5,24,,,,<SMS,8.,40>
000232     123     125     103       SMS:    .ASCII'SUCCESS!'
000235     103     105     123
000240     123     041
000242                               ERP:    QIOW$   IO.WVB,5,24,,,,<CMB,40.,40>
000264  000000  000000               IOST:   .WORD   0,0
000270                               CMB:    .BLKB   82.
```

```
TESTP - TEST PARSER      MACRO C1013  24-AUG-77 12:50  PAGE 2

                                          ; STATE TABLE FOR COMMAND
                                          ; UFD DK0:SIXNAM[100,100]/PRO=[RWED,RWED,RWED,RWED]/ALLOC=300
                                          ;
000412  000003                            START:  KEY.
000414     125     106     104                    .ASCIZ'UFD '
000417     040     000
                                                  .EVEN
000422  000000G                                   NULL.
000424  000430'                                   N1
000426  177777                                    -1
000430  000006                            N1:     ANY.
000432  000000G                                   NULL.
000434  000440'                                   N2
000436  177777                                    -1
000440  000006                            N2:     ANY.
000442  000000G                                   NULL.
000444  000450'                                   N3
000446  177777                                    -1
000450  000002                            N3:     NUMB.
000452  000000G                                   NULL.
000454  000466'                                   DEV1
000456  000005                                    MTCH.
000460  000000G                                   NULL.
000462  000466'                                   DEV1
000464  177777                                    -1
000466  000003                            DEV1:   KEY.
000470     072     000                            .BYTE   ':,0
000472  000000G                                   NULL.
000474  000500'                                   N4
000476  177777                                    -1
000500  000001                            N4:     ALPH.
000502  000406                                    406
000504  000000G                                   NULL.
000506  000512'                                   N5
000510  177777                                    -1
000512  000007                            N5:     SUBX.
000514  000704'                                   UIC
000516  000000G                                   NULL.
000520  000524'                                   OPTS
000522  177777                                    -1
000524  000004                            OPTS:   EOF.
000526  000000G                                   NULL.
000530  000764'                                   ENDIT
000532  000003                                    KEY.
000534     057     000                            .BYTE   '/,0
000536  000000G                                   NULL.
000540  000544'                                   N6
000542  177777                                    -1
000544  000003                            N6:     KEY.
000546     101     114     114                    .ASCIZ'ALLOC='
000551     117     103     075
000554     000
                                                  .EVEN
000556  000000G                                   NULL.
000560  000600'                                   ALC
000562  000003                                    KEY.
000564     120     122     117                    .ASCIZ'PRO='
```

```
TESTP - TEST PARSER    MACRO D1013  24-AUG-77 12:50  PAGE 2-1

000567     075     000
                                          .EVEN
000572  000000G                           NULL.
000574  000610'                           PRO
000576  177777                            -1
000600  000002                  ALC:      NUMB.
000602  000000G                           NULL.
000604  000524'                           OPTS
000606  177777                            -1
000610  000003                  PRO:      KEY.
000612     133     000                    .BYTE    '[,0
000614  000000G                           NULL.
000616  000622'                           SPRO
000620  177777                            -1
000622  000003                  SPRO:     KEY.
000624     135     000                    .BYTE    '],0
000626  000000G                           NULL.
000630  000524'                           OPTS
000632  000003                            KEY.
000634     054     000                    .BYTE    ',,0
000636  000000G                           NULL.
000640  000622'                           SPRO
000642  000003                            KEY.
000644     122     000                    .BYTE    'R,0
000646  000000G                           NULL.
000650  000622'                           SPRO
000652  000003                            KEY.
000654     127     000                    .BYTE    'W,0
000656  000000G                           NULL.
000660  000622'                           SPRO
000662  000003                            KEY.
000664     105     000                    .BYTE    'E,0
000666  000000G                           NULL.
000670  000622'                           SPRO
000672  000003                            KEY.
000674     104     000                    .BYTE    'D,0
000676  000000G                           NULL.
000700  000622'                           SPRO
000702  177777                            -1
000704  000003                  UIC:      KEY.
000706     133     000                    .BYTE    '[,0
000710  000000G                           NULL.
000712  000716'                           U1
000714  177777                            -1
000716  000002                  U1:       NUMB.
000720  000000G                           NULL.
000722  000726'                           U2
000724  177777                            -1
000726  000003                  U2:       KEY.
000730     054     000                    .BYTE    ',,0
000732  000000G                           NULL.
000734  000740'                           U3
000736  177777                            -1
000740  000002                  U3:       NUMB.
000742  000000G                           NULL.
000744  000750'                           U4
000746  177777                            -1
```

```
TESTP - TEST PARSEF      MACRO D1013  24-AUG-77 12:50  PAGE 2-2

    000750  000003                      U4:      KEY.
    000752     135     000                       .BYTE    '1,0
    000754  000000G                              NULL.
    000756  000762'                              U5
    000760  177777                               -1
    000762  000010                      U5:      ENDX.
    000764  000000                      ENDIT:   END.
                                        ;
            000000'                              .END     TESTP
```

```
TESTP - TEST PARSER      MACRO D1013  24-AUG-77 12:50  PAGE 2-3
SYMBOL TABLE

ALC      000600R         EXIT     000132R         N3       000450R         Q.IOFN= 000002         SUXMES  000210R
ALPH.  = 000001          IE.EOF= ****** GX        N4       000500R         Q.IOLU= 000004         TESTP   000000R
ANY    = 000006          IOST     000264R         N5       000512R         Q.IOPL= 000014         UIC     000704R
CMB      000270R         IO.RLB= ****** GX        N6       000544R         Q.IOPR= 000007         U1      000716R
CML      000166R         IO.WVB= ****** GX        OPTS     000524R         Q.IOSB= 000010         U2      000726R
DEV1     000466R         KEY.   = 000003          PARSE  = ****** GX       SMS      000232R       U3      000740R
ENDIT    000764R         MTCH.  = 000005          PRMPT    000140R         SPRO     000622R       U4      000750R
ENDX.  = 000010          NULL.  = ****** GX       PRO      000610R         START    000412R       U5      000762R
END.   = 000000          NUMB.  = 000002          PRPT     000162R         SUBX.  = 000007        $$$ARG= 000011
EOF.   = 000004          N1       000430R         Q.IOAE= 000012           SUX      000120R       $$$OST= 000014
ERP      000242R         N2       000440R         Q.IOEF= 000006

. ABS.   000000      000
         000766      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  1094 WORDS  ( 5 PAGES)
DYNAMIC MEMORY:  11961 WORDS  ( 46 PAGES)
ELAPSED TIME:  00:00:12
,LP:=DK1:TESTP.MAC
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50
TABLE OF CONTENTS
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50  PAGE 1

                .TITLE  PARSE - GSB PARSER SUBROUTINES
                ;
                ;
                ;
                ; A MACRO PARSER FOR TEXT STRINGS BASED LOOSELY ON .TPARS
                ; DESIGNED PRIMARILY FOR ANALYSIS OF COMMAND-LIKE STRUCTURES
                ; BUT INTENDED TO BE EXTENDABLE FOR ANY GRAMMATICAL ANALYSIS.
                ;
                ;
                ;
                ; WRITTEN BY G.S. BRIMBLE  28 SEPT 1976
                ;
                ; MODIFIED:
                ;
                ; 26/1/77       SYNTACTIC TYPE SYMBOLS CHANGED FOR UNIFORMITY
                ;               WITH NOVA PROGRAM.
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50  PAGE 2
INSTRUCTIONS FOR USE

                                .SBTTL  INSTRUCTIONS FOR USE.
                                ;
                                ; TO CAUSE ANALYSIS TO BE CARRIED OUT,A SUBROUTINE CALL
                                ;       JSR     PC,PARSE
                                ; IS USED REGISTER R0 WILL CONTAIN THE RESULT OF THE PARSE
                                ; ON RETURN - R0=0 IF SUCCESSFUL, =-2 IF FAILED
                                ; REGISTERS R3,R4,R5,R6 ARE UNAFFECTED BY THE PARSE.
                                ; R1 MUST POINT TO THE TEXT STRING TO BE ANALYSED.IF THE
                                ; PARSE FAILED,R1 WILL BE POINTING TO THE BYTE(S)
                                ; WHICH CAUSED THE FAILURE.
                                ; R2 MUST POINT TO THE ENTRY STATE OF A STATE TABLE
                                ; WHICH DEFINES THE STRUCTURE WHICH THE TEXT MUST MATCH.
                                ; THE STATE TABLE CONSISTS OF A SET OF TRANSITIONS
                                ; WHICH ARE SO ORDERED AS TO DEFINE THE ALLOWABLE STRUCTURE
                                ; OF THE TEXT.
                                ; THE TRANSITIONS CONSIST OF THREE ENTRIES:
                                ;       SYNTACTIC TYPE (PLUS ANY TRAILING ARGUMENTS)
                                ;       ACTION SUBROUTINE POINTER
                                ;       TARGET STATE POINTER
                                ;
                                ; THE ACTION SUBROUTINES ARE EXECUTED IF THE TRANSITION
                                ; SUCCEEDS,I.E. THE SYNTACTIC TYPE MATCHES. THEY ARE
                                ; CALLED FROM THE PARSER BY 'JSR PC,@POINTER' SO MUST
                                ; RETURN TO THE PARSER BY 'RTS PC'.REGISTERS R1,R2,R6 MUST BE
                                ; PRESERVED BY THE SUBROUTINES.
                                ; IF NO ACTION IS NEEDED WHEN A TRANSITION IS TAKEN,
                                ; A 'DO NOTHING' SUBROUTINE IS PROVIDED BY THE PARSER
                                ; CALLED 'NULL.'. THIS IS SPECIFIED AS THE ACTION
                                ; SUBROUTINE POINTER FOR NO ACTION.
                                ; THE TARGET STATE POINTER POINTS TO THE TRANSITION
                                ; TO BE ATTEMPTED IF THE CURRENT ONE SUCCEEDS. IF THIS
                                ; ONE FAILS,THE FOLLOWING ONE IS ATTEMPTED.THE END OF A
                                ; STATE IS INDICATED BY A PSEUDO-TYPE -1.IF THIS IS
                                ; ENCOUNTERED AS A TYPE THE PARSE FAILS.
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50  PAGE 3
SYNTACTIC TYPES

        .SBTTL  SYNTACTIC TYPES.
        ;
        ; THE SYNTACTIC TYPE ENTRY IN THE TRANSITION BLOCK
        ; MAY BE ONE OF THE FOLLOWING:
        ;
        ; ALPH.  DEFINED ALPH.=1
        ;        THIS MATCHES ALL FOLLOWING ALPHABETIC CHARACTERS
        ;        UNTIL A NON-ALPH. CHAR  IS FOUND. IT AMOUNTS TO
        ;        AN 'IGNORE TEXT' SPECIFICATION. THE ENTRY HAS A
        ;        TRAILING ONE-WORD ARGUMENT;THE LOWER BYTE IS A
        ;        MAXIMUM BYTE COUNT TO LIMIT THE AMOUNT OF TEXT
        ;        TO BE IGNORED,AND THE UPPER BYTE IS A FLAG
        ;        =0 IF SPACES(AND TABS) ARE TO BE TREATED AS ALPHA,
        ;        =1 IF SPACES CAN BE USED AS TERMINATORS.
        ;        THUS THIS TYPE ALWAYS SUCCEEDS UNLESS THE BYTE
        ;        COUNT IS EXCEEDED. NOTE THAT THIS TYPE SUCCEEDS IF
        ;        THERE ARE NO ALPHABETIC CHARS.
        ; NUMB.          DEFINED NUMB.=2
        ;        THIS MATCHES A DECIMAL INTEGER WHOSE VALUE IS
        ;        STORED IN THE PARSER-DEFINED LOCATION 'NUM.'.
        ;        CONVERSION STOPS AT FIRST NON-NUMERIC CHAR.
        ;        IT ALWAYS SUCCEEDS - IF NO NUMERALS,NUM.=0
        ; KEY.           KEY.=3
        ;        MATCHES ONE OR MORE ASCII CHARS WHICH FOLLOW THE
        ;        TYPE SPEC. AS TRAILING PARAMETERS. THESE KEY
        ;        CHARS MUST TERMINATE WITH A NULL BYTE. THEY ARE
        ;        EASILY DEFINED USING THE '.ASCIZ ' OPERATOR BUT
        ;        THIS MUST BE FOLLOWED BY '.EVEN' TO RETURN TO
        ;        WORD BOUNDARIES.
        ; EOF.           EOF.=4
        ;        MATCHES ESC,ALT MODE,FF,RUBOUT,CR(AND LF IF FOLLOWS).
        ;        ON ENTRY TO ACTION SUBROUTINE,R3 HAS THE CHAR.
        ; MTCH.          MTCH.=5
        ;        THIS CAUSES AN UNCONDITIONAL MATCH - ALWAYS SUCCEEDS.
        ;        IT FORCES ACTION AND/OR NEW STATE.
        ; ANY.           ANY.=6
        ;        PUTS NEXT CHAR. INTO PARSER-DEFINED LOCATION 'CHAR.'.
        ;        SUCCEEDS UNLESS NEXT CHAR IS EOF OR RUBOUT(377).
        ; SUBX.  SUBX.=7
        ;        THIS ASKS THE PARSER TO MATCH A SUB-EXPRESSION,
        ;        I.E. TO ATTEMPT TO MATCH A STRUCTURE DEFINED BY A
        ;        SUBSIDIARY STATE TABLE. THE TYPE ENTRY IS FOLLOWED
        ;        BY A TRAILING ARGUMENT OF A SINGLE WORD CONTAINING
        ;        THE ADDRESS OF THE ENTRY STATE OF THE SUBEXPRESSION.
        ;        THIS SUBEXPRESSION STATE TABLE MUST CONTAIN A
        ;        STATE WITH THE TYPE 'ENDX.' AS THE SUCCESS EXIT.
        ; ENDX.  ENDX.=10
        ;        SUCCESS EXIT FROM SUBEXPRESSION. THIS TYPE CAUSES
        ;        THE PARSER TO CARRY OUT THE SUCCESS ACTION AND
        ;        TARGET IN THE TRANSITION WHICH CALLED THE
        ;        SUBEXPRESSION. IT HAS NO ACTION OR TARGET ENTRIES
        ;        ASSOCIATED WITH IT.
        ; END.           END.=0
        ;        THIS TYPE IS A SPECIAL CASE. IT INDICATES SUCCESSFUL
        ;        PARSE AND CAUSES THE PARSER TO RETURN CONTROL TO
        ;        THE CALLING PROGRAM WITH THE SUCCESS CODE IN R0.
        ;        IT HAS NO ACTION OR TARGET ENTRIES.
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50  PAGE 4
DEFINITIONS

                                .SBTTL  DEFINITIONS
                                ;
        000000                          END.=0
        000001                          ALPH.=1
        000002                          NUMB.=2
        000003                          KEY.=3
        000004                          EOF.=4
        000005                          MTCH.=5
        000006                          ANY.=6
        000007                          SUBX.=7
        000010                          ENDX.=10
                                ;
                                ;
                                .SBTTL  CODE
                                ;
000000                          .PSECT  PARS$           ;MAKE PSECT TO AVOID SYMBOL CLASH.
                                ;
000000  010346                  PARSE::MOV      R3,-(SP)        ;SAVE WORKING REGS
000002  010446                          MOV     R4,-(SP)
000004  010546                          MOV     R5,-(SP)
000006  010167  000060          PARS$G: MOV     R1,TXTPT$       ;SAVE CURRENT TEXT PTR
000012  012200                          MOV     (R2)+,R0        ;GET TYPE
000014  006300                          ASL     R0              ;MAKE IT WORD ADDRESS
000016  000170  000050'                 JMP     @VECT$(R0)      ;VECTOR TO ITS ROUTINE
000022  004732                  SUXS$:  JSR     PC,@(R2)+       ;ON SUCCESS,DO ACTION
000024  011202                          MOV     (R2),R2         ;SET TARGET POINTER
000026  000167  177754                  JMP     PARS$G          ;REPEAT
000032  016701  000034          FAYL$:  MOV     TXTPT$,R1       ;RESTORE TEXT PTR TO AS BEFORE FAIL
000036  005732                          TST     @(R2)+          ;IGNORE ACTION
000040  005732                          TST     @(R2)+          ;AND TARGET
000042  000167  177740                  JMP     PARS$G          ;REPEAT.
                                ;
000046  000102'                         .WORD   XIT$
000050  000112'                 VECT$:  .WORD   END$
000052  000302'                         .WORD   ALPH$
000054  000360'                         .WORD   NUM$
000056  000234'                         .WORD   KEY$
000060  000124'                         .WORD   EOF$
000062  000120'                         .WORD   MTCH$
000064  000200'                         .WORD   ANY$
000066  000470'                         .WORD   SUBX$
000070  000524'                         .WORD   ENDX$
                                ;
000072  000000                  TXTPT$: .WORD   0
000074  000000                  CHAR.:: .WORD   0
000076  000000                  NUM.::  .WORD   0
000100  000207                  NULL.:: RTS     PC
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50  PAGE 5
CODE

                                  ; ROUTINES TO MATCH SYNTACTIC TYPES.
                                  ;
    000102  012605                XIT$:   MOV     (SP)+,R5        ;RESTORE REGS
    000104  012604                        MOV     (SP)+,R4
    000106  012603                        MOV     (SP)+,R3
    000110  000207                        RTS     PC
                                  ;
    000112  005000                END$:   CLR     R0              ;FLAG SUCCESS
    000114  000167  177762                JMP     XIT$
                                  ;
    000120  000167  177676        MTCH$:  JMP     SUXS$
                                  ;
    000124  012703  000164'       EOF$:   MOV     #40$,R3         ;POINT TO EOF TABLE
    000130  121123                30$:    CMPB    (R1),(R3)+
    000132  001405                        BEQ     10$             ;DOES IT MATCH?
    000134  020327  000176'               CMP     R3,#50$         ;NO - END OF TABLE?
    000140  101773                        BLOS    30$             ;NO - TRY ANOTHER
    000142  000167  177664                JMP     FAYL$           ;YES - NO MATCH SO FAIL
    000146  005201                10$:    INC     R1              ;PUSH POINTER IF MATCHES
    000150  121127  000012                CMPB    (R1),#12        ;FOLLOWED BY LF?
    000154  001001                        BNE     20$
    000156  005201                        INC     R1              ;IF SO, PUSH TEXT PTR
    000160  000167  177636        20$:    JMP     SUXS$           ;THEN SUCCEED
    000164  000015                40$:    .WORD   15
    000166  000377                        .WORD   377
    000170  000176                        .WORD   176
    000172  000033                        .WORD   33
    000174  000032                        .WORD   32
    000176  000014                50$:    .WORD   14
                                  ;
    000200  121127  000377        ANY$:   CMPB    (R1),#377       ;IF RUBOUT
    000204  001411                        BEQ     11$
    000206  121127  000032                CMPB    (R1),#32        ;OR END OF FILE,
    000212  001406                        BEQ     11$             ;THIS FAILS.
    000214  005067  177654                CLR     CHAR.           ;OTHERWISE CLEAR WORD TO
    000220  112167  177650                MOVB    (R1)+,CHAR.     ;STORE BYTE
    000224  000167  177572                JMP     SUXS$           ;AND SUCCEED.
    000230  000167  177576        11$:    JMP     FAYL$
                                  ;
    000234  105712                KEY$:   TSTB    (R2)            ;END OF KEYS?
    000236  001412                        BEQ     12$             ;THEN SUCCEED
    000240  122122                        CMPB    (R1)+,(R2)+     ;MATCH?
    000242  001774                        BEQ     KEY$            ;YES - DO ANOTHER
    000244  105722                22$:    TSTB    (R2)+           ;FAILED SO PUSH PTR OVER KEYS
    000246  001376                        BNE     22$             ;UNTIL ZERO IS FOUND
    000250  005202                        INC     R2              ;ALIGN TO WORD BOUNDARY
    000252  006002                        ROR     R2
    000254  000241                        CLC
    000256  006102                        ROL     R2
    000260  000167  177546                JMP     FAYL$
    000264  005202                12$:    INC     R2              ;SUCCESS - ALIGN TO WORD BOUNDARY
    000266  005202                        INC     R2
    000270  006002                        ROR     R2
    000272  000241                        CLC
    000274  006102                        ROL     R2
    000276  000167  177520                JMP     SUXS$
                                  ;
```

```
PARSE - GSB PARSER SUBROUTINES  MACRO D1013  24-AUG-77 12:50  PAGE 5-1
CODE

000302  112203                  ALPH$:  MOVB    (R2)+,R3        ;GET NO. OF CHARS
000304  112204                          MOVB    (R2)+,R4        ;AND SPACE FLAG
000306  005704                  43$:    TST     R4              ;COUNT SPACES AS ALPHA?
000310  001006                          BNE     13$
000312  121127  000040                  CMPB    (R1),#40        ;YES - ANY SPACES?
000316  001411                          BEQ     23$             ;YES - GO AGAIN
000320  121127  000011                  CMPB    (R1),#11        ;TRY TABS
000324  001406                          BEQ     23$
000326  121127  000101          13$:    CMPB    (R1),#'A        ;IS BYTE < A?
000332  103410                          BLO     33$             ;IF SO IT IS A BREAK CHAR
000334  121127  000132                  CMPB    (R1),#'Z        ;IS IT > Z?
000340  101005                          BHI     33$             ;IF SO BREAK
000342  005201                  23$:    INC     R1              ;MOVE TO NEXT BYTE
000344  005303                          DEC     R3              ;DONE ENOUGH TEXT?
000346  002357                          BGE     43$             ;NOT YET - DO NEXT BYTE
000350  000167  177456                  JMP     FAYL$           ;YES - FAIL
000354  000167  177442          33$:    JMP     SUXS$           ;FOUND BREAK SO SUCCEED.
                                ;
000360  005003                  NUM$:   CLR     R3              ;CLEAR SIGN FLAG
000362  005067  177510                  CLR     NUM.            ;AND SUM WORD
000366  121127  000053                  CMPB    (R1),#'+        ;PLUS SIGN?
000372  001404                          BEQ     14$
000374  121127  000055                  CMPB    (R1),#'-        ;MINUS?
000400  001001                          BNE     14$             ;NO SIGN SO ASSUME +
000402  005203                          INC     R3              ;FLAG MINUS
000404  121127  000060          14$:    CMPB    (R1),#'0        ;IF CHAR < 0
000410  103421                          BLO     24$
000412  121127  000071                  CMPB    (R1),#'9        ;OR > 9
000416  101016                          BHI     24$             ;THEN BREAK AND FINISH
000420  112104                          MOVB    (R1)+,R4        ;OTHERWISE GET NUMERAL
000422  162704  000060                  SUB     #'0,R4          ;MAKE A NUMBER
000426  016700  177444                  MOV     NUM.,R0
000432  006300                          ASL     R0              ;MULT. BY 10 THE HARD WAY
000434  006300                          ASL     R0              ;SO THIS CAN RUN ON PDP11/10
000436  060067  177434                  ADD     R0,NUM.  ;2*(NO.+(2*(2*NO.)))=10*NO.
000442  006367  177430                  ASL     NUM.
000446  060467  177424                  ADD     R4,NUM.  ;ADD IN NEW DIGIT
000452  000754                          BR      14$             ;GET ANOTHER DIGIT
000454  005703                  24$:    TST     R3              ;BREAK SO APPLY SIGN
000456  001402                          BEQ     34$
000460  005467  177412                  NEG     NUM.
000464  000167  177332          34$:    JMP     SUXS$
                                ;
000470  010246                  SUBX$:  MOV     R2,-(SP)        ;SAVE CURRENT STATE PTR
000472  011202                          MOV     (R2),R2         ;GET NEW ONE
000474  012767  000506' 177344          MOV     #SUBXT$,VECT$-2 ;ALTER ERROR PTR IN TABLE
000502  000167  177300                  JMP     PARS$G          ;SO IF SUBEX FAILS IT COMES HERE.
000506  012602                  SUBXT$: MOV     (SP)+,R2        ;RESTORE STATE PTR
000510  005722                          TST     (R2)+           ;SKIP TRAILING ARGUMENT
000512  012767  000102' 177326          MOV     #XIT$,VECT$-2   ;RESTORE ERROR POINTER
000520  000167  177306                  JMP     FAYL$           ;AND TREAT AS NORMAL TRANS FAIL
000524  012602                  ENDX$:  MOV     (SP)+,R2        ;SUBEX HAS SUCCEEDED
000526  005722                          TST     (R2)+           ;SO RESTORE AS ABOVE
000530  012767  000102' 177310          MOV     #XIT$,VECT$-2
000536  000167  177260                  JMP     SUXS$           ;AND SUCCEED.
                                ;
        000001                          .END
```

```
PARSE - CSE PARSER SUBROUTINES  MACRO 01013  24-AUG-77 12:50  PAGE 5-2
SYMBOL TABLE

ALPH$    000702R     002 ENDX  = 000010          KEY$     000234R     002 NUM$     000360R     002 SUBX  = 000007
ALPH   = 000001          END$    000112R     002 KEY   = 000003           NUM.     000076RG    002 SUXS$    000022R     002
ANY$     000200R     002 END   = 000000          MTCH$    000120R     002 PARSE    000000RG    002 TXTPT$   000072R     002
ANY    = 000006          EOF$    000124R     002 MTCH  = 000005           PARS$G   000006R     002 VECT$    000050R     002
CHAR     000074RG    002 EOF   = 000004          NULL.    000100RG    002 SUBXT$   000506R     002 XIT$     000102R     002
ENDX$    000524R     002 FAYL$   000032R     002 NUMB  = 000002           SUBX$    000470R     002

  ABS    000000      000
         000000      001
PARS$    000542      002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  261 WORDS  ( 2 PAGES)
DYNAMIC MEMORY:  11961 WORDS  ( 46 PAGES)
ELAPSED TIME:  00:00:11
,LP:=DK1:PARSER.MAC
```

APPENDIX IV

NOVA PARSER PROGRAM LISTING

```
                        .TITL TESTP
                        ; ILLUSTRATION OF PARSER USE.
                        ; G. BRIMBLE  4/3/77
                        ;
      000001                    .TXTM   1
                        ;
                        ;  DEFINE GLOBAL SYMBOLS IF NEEDED BY OTHER TASKS
                        ;
                                .ENT    CHAR.,NUM.,PARSI
                                .ENT    TESTP
                        ;
                                .ZREL
00000-000377  IDMSK:   377                  ;USER MUST DEFINE THIS ON P.ZERO
00001-000000  CHAR.:   0                    ;AND THIS AND THE NEXT ONE
00002-000000  NUM.:    0
00003-000367' PARSI:   PARSE
              ;
                       .NREL
00000'020444  TESTP:   LDA     0,TTIPT              ;OPEN CH0 TO TTI
00001'006017           .SYSTM
00002'014000           .OPEN   0
00003'063077           HALT                         ;ERROR
00004'020444           LDA     0,TTOPT              ;OPEN CH1 TO TTO
00005'006017           .SYSTM
00006'014001           .OPEN   1
00007'063077           HALT
00010'020444  LOOPP:   LDA     0,PRMPT
00011'006017           .SYSTM
00012'017001           .WRL    1                    ;INDICATE READY
00013'000775           JMP     .-3                  ;ERROR RETURN
00014'020444           LDA     0,CML
00015'006017           .SYSTM
00016'015400           .RDL    0                    ;GET COMMAND
00017'000775           JMP     .-3                  ;ERROR
00020'024440           LDA     1,CML
00021'125220           MOVZR   1,1                  ;WORD PTR TO TEXT
00022'030557           LDA     2,STRPT              ;PREPARE TO PARSE
00023'006003-          JSR     @PARSI               ;DO IT
00024'101005           MOV     0,0,SNR              ;WAS IT OK?
00025'000407           JMP     GOODP                ;YES
00026'020321           LDA     0,TXPT               ;NO-SAY WHY
00027'101120           MOVZL   0,0                  ;BYTE PTR
00030'006017           .SYSTM
00031'017001           .WRL    1
00032'000756           JMP     LOOPP                ;ERROR
00033'000755           JMP     LOOPP
00034'020402  GOODP:   LDA     0,SUXM
00035'000773           JMP     .-5                  ;PRINT "SUCCESS!"
              ;
00036'000076" SUXM:    .+1*2
00037'051525           .TXT/SU
00040'041503  CC
00041'042523  ES
00042'051441  S!
00043'006400  <15>/
00044'000112" TTIPT:   .+1*2
00045'022124           .TXT/$T
00046'052111  TI
00047'000000  /
00050'000122" TTOPT:   .+1*2
```

```
---
00051'022124          .TXT/ST
00052'052117  TO
00053'000000  /
00054'000132" PRMPT:  .+1*2
00055'052123          .TXT/TS
00056'052076  T>
00057'000000  /
00060'000142" CML:    .+1*2
      000120          .BLK    80.
              ;
              ;  TRANSITION TABLE
              ;
00201'000202' STRPT:  START
00202'000003  START:  KEY.
00203'052506          .TXT/UF
00204'042040  D
00205'000000  /
00206'000477'         NULL.
00207'000211'         N1
00210'177777          -1
00211'000006  N1:     ANY.
00212'000477'         NULL.
00213'000215'         N2
00214'177777          -1
00215'000006  N2:     ANY.
00216'000477'         NULL.
00217'000221'         N3
00220'177777          -1
00221'000002  N3:     NUMB.
00222'000477'         NULL.
00223'000230'         DEV1
00224'000005          MTCH.
00225'000477'         NULL.
00226'000230'         DEV1
00227'177777          -1
00230'000003  DEV1:   KEY.
00231'035000          256.*":
00232'000477'         NULL.
00233'000235'         N4
00234'177777          -1
00235'000001  N4:     ALPH.
00236'000000          0
00237'000477'         NULL.
00240'000242'         N5
00241'177777          -1
00242'000007  N5:     SUBX.
00243'000336'         UIC
00244'000477'         NULL.
00245'000247'         OPTS
00246'177777          -1
00247'000004  OPTS:   EOF.
00250'000477'         NULL.
00251'000366'         ENDIT
00252'000003          KEY.
00253'027400          256.*"/
00254'000477'         NULL.
00255'000256'         N6
00256'000003  N6:     KEY.
00257'040514          .TXT/AL
00260'046117  LO
```

```
---
 00261'041475  C=
 00262'000000  /
 00263'030477'           NULL.
 00264'030274'           ALC
 00265'000003            KEY.
 00266'050122            .TXT/PR
 00267'047475  O=
 00270'030000  /
 00271'000477'           NULL.
 00272'300300'           PRO
 00273'177777            -1
 00274'030002  ALC:      NUMB.
 00275'000477'           NULL.
 00276'030247'           OPTS
 00277'177777            -1
 00300'000003  PRO:      KEY.
 00301'055400            .TXT/[/
 00302'000477'           NULL.
 00303'030305'           SPRO
 00304'177777            -1
 00305'000003  SPRO:     KEY.
 00306'056400            .TXT/]/
 00307'030477'           NULL.
 00310'030247'           OPTS
 00311'000003            KEY.
 00312'026000            .TXT/,/
 00313'000477'           NULL.
 00314'300305'           SPRO
 00315'000003            KEY.
 00316'051000            .TXT/R/
 00317'000477'           NULL.
 00320'030305'           SPRO
 00321'030003            KEY.
 00322'053400            .TXT/W/
 00323'030477'           NULL.
 00324'030305'           SPRO
 00325'030003            KEY.
 00326'042400            .TXT/E/
 00327'000477'           NULL.
 00330'030305'           SPRO
 00331'030003            KEY.
 00332'042000            .TXT/D/
 00333'030477'           NULL.
 00334'000305'           SPRO
 00335'177777            -1
 00336'000003  UIC:      KEY.
 00337'055400            .TXT/[/
 00340'030477'           NULL.
 00341'000343'           U1
 00342'177777            -1
 00343'030002  U1:       NUMB.
 00344'030477'           NULL.
 00345'030347'           U2
 00346'177777            -1
 00347'030003  U2:       KEY.
 00350'026000            .TXT/,/
 00351'030477'           NULL.
 00352'030354'           U3
 00353'177777            -1
 00354'030002  U3:       NUMB.
```

```
---
00355'000477'           NULL.
00356'000360'           U4
00357'177777            -1
00360'000003  U4:       KEY.
00361'056400            .TXT/1/
00362'000477'           NULL.
00363'030365'           U5
00364'177777            -1
00365'000010  U5:       ENDX.
00366'000000  ENDIT:    END.
              ;
              ; PARSER MODULE
              ;
              ; DEFINITIONS
       000020           STPT=20
       000021           TXPT=21
       000030           STACK=30
       000000           END.=0
       000001           ALPH.=1
       000002           NUMB.=2
       000003           KEY.=3
       000004           EOF.=4
       000005           MTCH.=5
       000006           ANY.=6
       000007           SUBX.=7
       000010           ENDX.=10
              ;
       000001           .TXTM   1 .
              ;
00367'044021  PARSE:    STA     1,TXPT   ;SET UP AUTO-INC PTRS
00370'014021            DSZ     TXPT     ;DECR FOR AUTOINC PTR
00371'044462            STA     1,TSAVE
00372'014461            DSZ     TSAVE
00373'050020            STA     2,STPT
00374'014020            DSZ     STPT
00375'054416            STA     3,VECTAB+1  ;SET UP ERROR RETURN
00376'126400            SUB     1,1
00377'044452            STA     1,DTEM   ;CLEAR OLD BYTE
00400'024411            LDA     1,STKPT
00401'044030            STA     1,STACK
00402'020447  PARSG:    LDA     0,DTEM   ;SAVE PRESENT BYTE
00403'040447            STA     0,DSAVE
00404'024406            LDA     1,VECTAB
00405'032020            LDA     2,@STPT  ;GET A TRANSITION TYPE
00406'141000            MOV     2,0      ;IF -1 WILL BE ERROR CODE
00407'133000            ADD     1,2      ;MAKE AN ADDRESS
00410'003000            JMP     @0,2     ;VECTOR TO MATCHING TYPE
              ;
00411'000736' STKPT:    STAK
00412'000414' VECTAB:   .+2
00413'000000            0
00414'000425'           END
00415'000500'           ALPH
00416'000621'           DNUMB
00417'000551'           KEYWD
00420'000576'           EOF
00421'000575'           LAMDA
00422'000615'           ANY
00423'000674'           SUBX
00424'000706'           ENDX
```

```
---
                ;
00425'102400  END:    SUB     0,0       ;SUCCESSFUL PARSE
00426'002765          JMP     @VECTAB+1  ;RETURN FROM PARSE
                ;
                ; NXDAT GETS NEXT DATA BYTE IGNORING SPACES AND NULLS
                ; RETURNS BYTE IN AC1
                ;
00427'000000          0
00430'054777  NXDAT:  STA     3,.-1
00431'024420          LDA     1,DTEM
00432'044416          STA     1,LASTD
00433'176400          SUB     3,3
00434'054415          STA     3,DTEM   ;GET,THEN CLEAR,DTEM.
00435'125014          SKPZR   1,1      ;IS IT ZERO?
00436'002771          JMP     @NXDAT-1  ;NO-LEGIT. CHAR
00437'026321          LDA     1,@TXPT  ;YES-GET A NEW WORD,
00440'034000-         LDA     3,IDMSK
00441'137400          AND     1,3      ;MASK OFF LS BYTE
00442'054407          STA     3,DTEM   ;AND SAVE IT
00443'125300          MOVS    1,1      ;SWAP MS BYTE LOW
00444'034000-         LDA     3,IDMSK
00445'167405          AND     3,1,SNR  ;MASK IT OFF AND TEST FOR NULL
00446'000763          JMP     NXDAT+1  ;IF NULL,GET ANOTHER.
00447'002760          JMP     @NXDAT-1  ;ELSE AC1 IS NEXT BYTE
00450'000000  LASTD:  0
                ;
                ; SUCCESS ROUTINES: SUXEX(SUXX)RESTORES THE LAST BYTE
                ; TAKEN BY NXDAT,SUXS(SUX) DOES NOT. BOTH CAUSE
                ; EXECUTION OF THE ACTION ROUTINE AND BRANCH TO TRANS.
                ;
                ;
                ; STORAGE COMMON TO NXD AND SUX,FALE
                ;
00451'000000  DTEM:   0
00452'000000  DSAVE:  0
00453'000000  TSAVE:  0
                ;
00454'034774  SUXEX:  LDA     3,LASTD  ;RESTORE LAST BYTE TAKEN.

00455'054774          STA     3,DTEM
00456'175005          MOV     3,3,SNR
00457'014021          DSZ     TXPT     ;IF DTEM=0,PUSH THE PTR BACK
00460'034021  SUXS:   LDA     3,TXPT   ;IF NEXT MATCH FAILS,
00461'054772          STA     3,TSAVE  ;RETURN TO HERE.
00462'036020          LDA     3,@STPT
00463'005400          JSR     0,3      ;DO ACTION
00464'022020          LDA     0,@STPT  ;GET ADDR OF NEXT TRANS
00465'040020          STA     0,STPT   ;AND SET IT UP
00466'014020          DSZ     STPT     ;ALLOW FOR AUTO INC
00467'002531          JMP     @PARET   ;RETURN TO PARSE
00470'034762  FAYL:   LDA     3,DSAVE  ;NO MATCH- RESTORE TEXT POINTERS
00471'054760          STA     3,DTEM   ;TO AS BEFORE ATTEMPTED MATCH.
00472'034761          LDA     3,TSAVE
00473'054021          STA     3,TXPT
00474'036020          LDA     3,@STPT
00475'036020          LDA     3,@STPT  ;IGNORE ACTION AND TRANS
00476'002522          JMP     @PARET   ;GO BACK TO PARSE
                ;
                ; NULL ACTION ROUTINE FOR OTHER TASKS
00477'001400  NULL.:  JMP     0,3
                ;
```

```
                    ; TYPE MATCHING ROUTINES ENTERED BY VECTOR TABLE.
                    ;
00500'022020  ALPH:   LDA    0,@STPT ;GET ARGUMENT
00501'024000-         LDA    1,IDMSK
00502'107400          AND    0,1     ;MASK OFF BYTE COUNT
00503'044432          STA    1,BCNT
00504'101300          MOVS   0,0
00505'024000-         LDA    1,IDMSK
00506'123400          AND    1,0     ;GET SPACE FLAG
00507'000404          JMP    .+4
00510'014425  STRNG:  DSZ    BCNT    ;DONE BYTE COUNT?
00511'000402          JMP    .+2
00512'000756          JMP    FAYL    ;IF SO,FAILURE
00513'006456          JSR    @NXD    ;GET A BYTE SAVING AC0
00514'034456          LDA    3,ASCA
00515'136032          SKPGE  1,3
00516'000404          JMP    ENDST   ;FINISH IF <A
00517'034454          LDA    3,ASCZ
00520'136432          SKPGT  1,3     ;FINISH IF >Z
00521'000767          JMP    STRNG   ;CHAR WAS ALPH. SO DO ANOTHER
00522'101004  ENDST:  MOV    0,0,SZR ;ARE SPACES ALPHA?
00523'002443          JMP    @SUXX   ;RESTORE UNMATCHED BYTE AND END
00524'034407          LDA    3,ASCSP ;YES-IS THIS SPACE OR TAB?
00525'166415          SKPNE  3,1
00526'000762          JMP    STRNG   ;IF SO,GET ANOTHER ALPH
00527'034405          LDA    3,ASCTB
00530'166415          SKPNE  3,1
00531'000757          JMP    STRNG
00532'002434          JMP    @SUXX   ;NOT ALPHA SO SUCCEED
00533'000040  ASCSP:  40
00534'000011  ASCTB:  11
00535'000000  BCNT:   0
              ;
00536'020436  NXKEY:  LDA    0,KTEM  ;ALGOR. SIM TO NXD GETS
00537'152400          SUB    2,2     ;NEXT KEY BYTE IN AC0.
00540'050434          STA    2,KTEM
00541'101014          SKPZR  0,0
00542'000404          JMP    .+4     ;MATCH A BYTE
00543'022020          LDA    0,@STPT
00544'040430          STA    0,KTEM  ;IF NULL,GET AND SAVE ANOTHER
00545'101300          MOVS   0,0     ;SWAP MS BYTE LOW
00546'030000-         LDA    2,IDMSK ;MASK OFF THE NEXT BYTE
00547'143400          AND    2,0
00550'001400          JMP    0,3     ;CARRY BACK IN AC0
00551'004765 .KEYWD:  JSR    NXKEY   ;GET A KEY
00552'101004          MOV    0,0,SZR ;IS IT END(NULL)?
00553'000403          JMP    MCHKY   ;NO-MATCH IT
00554'040420          STA    0,KTEM  ;YES-RESET STORE
00555'002412          JMP    @SUX    ;IF NULL, SUCCESS!
00556'006413  MCHKY:  JSR    @NXD    ;GET A DATA BYTE
00557'106415          SKPNE  0,1     ;MATCH?
00560'000771          JMP    KEYWD   ;YES-DO ANOTHER
00561'004755          JSR    NXKEY   ;NO-FAIL MATCH
00562'101004          MOV    0,0,SZR ;IGNORE KEYS UNTIL NULL
00563'000776          JMP    .-2
00564'040410          STA    0,KTEM  ;CLEAR STORE
00565'002403          JMP    @FALE   ;FAIL MATCH
              ;
              ; INDIRECT POINTERS AND CONSTANTS
00566'000454' SUXX:   SUXEX
```

```
---
00567'000460' SUX:     SUXS
00570'000470' FALE:    FAYL
00571'000430' NXD:     NXDAT
00572'000101  ASCA:    "A
00573'000132  ASCZ:    "Z
00574'000000  KTEM:    0
              ;
              ; REST OF MATCH ROUTINES
              ;
00575'002772  LAMDA:   JMP     @SUX     ;ALWAYS SUCCEED
              ;
00576'006773  EOF:     JSR     @NXD     ;GET NEXT BYTE
00577'030410           LDA     2,EOFT
00600'151400  ELOOP:   INC     2,2
00601'021000           LDA     0,0,2
00602'100015           SKPNM   0,0      ;END OF POSSIBILITIES?
00603'002765           JMP     @FALE    ;THEN FAIL
00604'106415           SKPNE   0,1      ;WAS IT A MATCH?
00605'002762           JMP     @SUX     ;YES
00606'000772           JMP     ELOOP    ;NO
00607'000607' EOFT:    .
00610'000015           15               ;CR
00611'000033           33               ;ESC
00612'000176           176              ;ALT MODE
00613'000014           14               ;FF
00614'177777           -1               ;END OF LIST
              ;
00615'006754  ANY:     JSR     @NXD     ;GET A BYTE
00616'044001-          STA     1,CHAR.  ;AND STORE IT
00617'002750           JMP     @SUX     ;SUCCESSFUL EXIT.
              ;
00620'000402' PARET:   PARSG
              ;
00621'102400  DNUMB:   SUB     0,0      ;BORROWED FROM .DBIN
00622'040444           STA     0,.EC10  ;CLEAR SIGN WORD
00623'040444           STA     0,.EC11  ;CLEAR SUM WORD
00624'006745           JSR     @.EC40   ; GET A CHARACTER
00625'121000           MOV     1,0
00626'024442           LDA     1,.EC20  ; TEST FOR "+"
00627'106405           SUB     0,1,SNR
00630'000405           JMP     .EC97    ; YES
00631'024440           LDA     1,.EC21  ; NO, TEST FOR "-"
00632'106404           SUB     0,1,SZR
00633'000404           JMP     .EC96    ; NO EXPLICIT SIGN
00634'010432           ISZ     .EC10    ; SET FLAG WORD FOR NEGATIVE
                                        ; NUMBER
00635'006734  .EC97:   JSR     @.EC40   ; GET ANOTHER CHARACTER
00636'121000           MOV     1,0
00637'024433  .EC96:   LDA     1,.EC22  ; ASCII "0"
00640'030433           LDA     2,.EC23  ; ASCII "9"
00641'142033           ADCZ#   2,0,SNC  ; SKIP IF > 9
00642'106032           ADCZ#   0,1,SZC  ; SKIP IF >= 0
00643'000406           JMP     .EC95    ; NOT A DIGIT, THERFORE A BREAK
00644'122400           SUB     1,0      ; REDUCE DIGIT TO 0-9 BINARY
00645'024422           LDA     1,.EC11  ; SUM WORD
00646'004412           JSR     .EC50    ; MULTIPLY BY 10 AND ADD
00647'044420           STA     1,.EC11  ; SAVE SUM
00650'000765           JMP     .EC97    ; GET NEXT CHARACTER
00651'024416  .EC95:   LDA     1,.EC11  ; RESULT TO AC1
00652'125120           MOVZL   1,1
```

```
---
00653'014413            DSZ     .EC10   ; TEST SIGN
00654'125221            MOVZR   1,1,SKP ; POSITIVE
00655'124640            NEGOR   1,1     ; NEGATIVE
00656'044002-           STA     1,NUM.
00657'002707            JMP     @SUXX   ;RESTORE BREAK CHAR.
              ; ROUTINE TO MULTIPLY AC1 BY 10 AND ADD AC0
00660'131120  .EC50:    MOVZL   1,2     ; N*2
00661'151120            MOVZL   2,2     ; N*4
00662'147000            ADD     2,1     ; N*5
00663'125120            MOVZL   1,1     ; N*5*2 = N*10
00664'107000            ADD     0,1     ; ADD AC0
00665'001400            JMP     0,3     ; SUCCESS RETURN
              ;
00666'000000  .EC10:    0               ; FLAG WORD FOR SIGN OF RESULT
00667'000000  .EC11:    0               ; RUNNING SUM WORD
00670'000053  .EC20:    "+              ; ASCII "+"
00671'000055  .EC21:    "-              ; ASCII "-"
00672'000060  .EC22:    "0              ; ASCII "0"
00673'000071  .EC23:    "9              ; ASCII "9"
      000571' .EC40=NXD                 ; ADDRESS OF GET CHARACTER
                                        ; ROUTINE
              ;
              ;
              ;
00674'026020  SUBX:     LDA     1,@STPT         ;GET NEW STATE PTR
00675'030020            LDA     2,STPT          ;STACK CURRENT STATE PTR
00676'052030            STA     2,@STACK
00677'044020            STA     1,STPT
00700'014020            DSZ     STPT            ;SET NEW STATE PTR
00701'026422            LDA     1,@VCTP
00702'046030            STA     1,@STACK        ;SAVE OLD ERROR RETURN
00703'024417            LDA     1,SXTP
00704'046417            STA     1,@VCTP         ;NOW ERRORS COME HERE
00705'002713            JMP     @PARET          ;TRY TO PARSE SUBEX
              ;
00706'126400  ENDX:     SUB     1,1             ;FLAG SUCCESS
00707'034030  SUBXT:    LDA     3,STACK
00710'021400            LDA     0,0,3           ;GET OLD ERROR RETN
00711'042412            STA     0,@VCTP
00712'175400            INC     3,3
00713'031400            LDA     2,0,3           ;GET OLD STATE PTR
00714'050020            STA     2,STPT
00715'175400            INC     3,3
00716'054030            STA     3,STACK         ;TIDY STACK
00717'125004            MOV     1,1,SZR         ;TEST FOR FAIL
00720'002650            JMP     @FALE
00721'002646            JMP     @SUX
00722'000707' SXTP:     SUBXT
00723'000413' VCTP:     VECTAB+1
      000012            .BLK    10.
      000736'           STAK=.
              ;
              ;
      000000'           .END    TESTP
```

```
---
ALC     000274'
ALPH    000500'
ALPH.   000001
ANY     000615'
ANY.    000006
ASCA    000572'
ASCSP   000533'
ASCTB   000534'
ASCZ    000573'
BCNT    000535'
CHAR.   000001-
CM      000000U
CML     000060'
DD      000000U
DEVI    000230'
DNUMB   000621'
DSAVE   000452'
DTEM    000451'
DZ      000000U
ELOOP   000600'
END     000425'
ENDIT   000366'
ENDST   000522'
ENDX    000706'
ENDX.   000010
END.    000000
EOF     000576'
EOFT    000607'
EOF.    000004
FALE    000570'
FAYL    000470'
GOODP   000034'
IDMSK   000000-
KEYWD   000551'
KEY.    000003
KE.     000000U
KTEM    000574'
LAMDA   000575'
LASTD   000450'
LOOPP   000010'
MCHKY   000556'
MTCH.   000005
N1      000211'
N2      000215'
N3      000221'
N4      000235'
N5      000242'
N6      000256'
NULL.   000477'
NUMB.   000002
NUM.    000002-
NXD     000571'
NXDAT   000430'
NXKEY   000536'
OPTS    000247'
ORE     000000U
PARET   000620'
PARSE   000367'
PARSG   000402'
PARSI   000003-
```

```
---
PRMPT   000054'
PRO     000300'
SPRO    000305'
STACK   000030
STAK    000736'
START   000202'
STKPT   000411'
STPT    000020
STRNG   000510'
STRPT   000201'
SUBX    000674'
SUBXT   000707'
SUBX.   000007
SUX     000567'
SUXEX   000454'
SUXM    000036'
SUXS    000460'
S
 UXX    000566'
SXTP    000722'
TESTP   000000'
TSAVE   000453'
TTIPT   000044'
TTOPT   000050'
TXPT    000021
U1      000343'
U2      000347'
U3      000354'
U4      000360'
U5      000365'
UIC     000336'
ULL.    000000U
VCTP    000723'
VECTA   000412'
.EC10   000666'
.EC11   000667'
.EC20   000670'
.EC21   000671'
.EC22   000672'
.EC23   000673'
.EC40   000571'
.EC50   000660'
.EC95   000651'
.EC96   000637'
.EC97   000635'
```

APPENDIX V

GLOSSARY OF TERMS

BNF:	Backus-Naur Form, a method to describe languages.
FINITE-STATE GRAMMAR:	A set of rules which define a language in which there are a finite number of combinations of elements.
FINITE-STATE PARSER:	A mechanism to determine whether input data conforms with a given finite-state grammar.
FORMALISM:	Formal structure.
GRAMMAR:	Rules defining a language.
META-COMPONENT:	Parts of a language being defined in BNF.
META-LANGUAGE:	A language which can be used to describe languages.
OBJECT:	A mark or symbol which is a basic undefinable element of the language being described; in particular, a string of letters may become a basic symbol independent of the individual letters from which it is composed.
PARSER:	A mechanism for deciding whether input data conforms to a particular grammar.
PHRASE-STRUCTURE GRAMMAR:	A set of rules which define a language by building more complex structures from aggregations of elements and other structures.
PSEUDO-OBJECT:	A meta-component which, although built up from objects, is a useful conceptual unit and can be treated as though it was itself an object.
REGULAR GRAMMAR:	A finite-state grammar.
SEMANTICS:	The meaning of constructs in a grammar; their significance in the context in which they are used.
STATE TABLE:	A data table which can direct a finite-state parser by specifying how the elements of the language must be combined.
SYNTACTIC TYPE:	An object or pseudo-object.
SYNTAX:	The arrangement of elements of a language according to its grammar.
TRANSITION TABLE:	State table

```
1 : <letter>          ; meta-component
      2               ; target
    <anything else>   ; next meta-component
      5               ; its target
2 :   =               ; first meta-component of
      3               ; new state
    <anything else>
      5
3 : <letter>
      4
    <anything else>
      5
4 : <operator>
      3
    <null>
      6
    <anything else>
      5
5 : ERROR
6 : SUCCESS
```

Figure 1(a)

```
1 : <letter>   ; meta-component
      2        ; target
     -1        ; end of state
2 :   =
      3
     -1
3 : <letter>
      4
     -1
4 : <operator>
      3
    <null>
      6
     -1
6 : SUCCESS
```

Figure 1(b)

Figure 1. Evolution of list structure

DISTRIBUTION

	Copy No.
EXTERNAL	
Chief Defence Scientist	1
Executive Controller, Australian Defence Scientific Service	2
Superintendent, Defence Science Administration	3
Assistant Secretary, Defence Information Services (for microfilming)	4
Defence Library, Campbell Park	5
Library, Aeronautical Research Laboratories	6
Library, Materials Research Laboratories	7
Director, Joint Intelligence Organisation (DDSTI)	8
Dr R. Vincent, Physics Department, University of Adelaide	9
Director, Mawson Institute for Antarctic Research	10
Documents distributed through ASDIS	
United Kingdom, for Ministry of Defence, Defence Research Information Centre (DRIC)	11
United States, for Department of Defense, Defense Documentation Center	12 - 24
Canada, for Ministry of Defence, Defence Science Information Service	25
New Zealand, for Ministry of Defence	26
Australian National Library	27
INTERNAL	
Director	28
Chief Superintendent, Applied Physics Wing	29
Superintendent, Electronics Division	30
Principal Officer, Cybernetic Electronics Group	31
Principal Officer, Radar and Electronic Tracking Group	32
Principal Officer, Jindalee Group	33
Principal Officer, Radio Group	34
Principal Officer, Electro Optics Group	35
Principal Officer, Laser Group	36
Principal Officer, Underwater Detection Group	37
Principal Officer, Systems Integration Group	38
Principal Officer, Communications Technology Group	39
Principal Officer, Central Test House	40
Principal Officer, Computer Aided Processes Group	41
Principal Officer, Microelectronics and Computer Applications Group	42
Principal Officer, Computing Services	43
Principal Officer, Planning and Data Analysis Group	44

	Copy No.
Mr. P.C. Drewer, Cybernetic Electronics Group	45
Mr. J.C. Mackenzie, Cybernetic Electronics Group	46
Mr. W.J. Rundle, Cybernetic Electronics Group	47
Mr. J.F. Duffield, Cybernetic Electronics Group	48
Author	49 - 50
Library, W.R.E.	51 - 52
Spares	53 - 69