SARPSIS: Syntax Analyzer, Recognizer, Parser and Semantic Interpretation System

November 15, 1967
CGTM # 34

The attached notes describe a system for describing and implementing specialized interpretive languages using PL/1. The work was supported by an NSF grant to E.B. Parker, Department of Communication, Stanford University.

The notes were written assuming the reader has a limited knowledge of phrase structured languages. An application of SARPSIS forms another report (the SPIRES Scope Demonstration System).

J.E. George

Computer Notes #25 September 29, 1967
SPIRES
(Stanford Physics Information REtrieval System)
JEG

SARPSIS: SYNTAX ANALYZER, RECOGNIZER, PARSER AND SEMANTIC INTERPRETATION SYSTEM

A general system for defining and implementing specialized computer languages

An information retrieval search language was required for the Stanford Physics Information Retrieval System (SPIRES). The objectives were:

1. A concise representation for describing the complete syntax and associated semantics;
2. An easy method of changing the syntax and/or semantics (with special emphasis on being able to easily expand the language to include additional types of requests);
3. The language should be unambiguous;
4. Recursive definitions should be possible within the syntax.

The method of Simple Precedence Grammars as reported by Wirth and Weber (1966a, 1966b) was chosen to accomplish these objectives. In this system, the grammar which generates any sentence in a particular language is written as a production grammar in Backus Normal Form (BNF). The syntax is first analyzed to check for simple precedence satisfaction and this analyzer produces output tables which are then used to parse sentences of the language defined by the analyzed syntax.

The representation of a language as being defined by a production grammar in BNF is concise and only ambiguous if the grammar itself is ambiguous. Simple precedence grammars are by definition unambiguous, hence any language which can be defined by a simple precedence grammar will be unambiguous. Further, it permits recursive definitions. Thus most of the objectives will be fulfilled by this technique and the rest can be by proper implementation.

The problem in most languages occurs in the parsing (or reduction) stage as to whether to use a left to right, a right to left, or a middle out parse. The parsing becomes more complicated if local ambiguities exist,

but is solvable (e.g., COGENT, Reynolds, 1965). However, if the language has global ambiguities then it cannot be resolved.

My viewpoint is that many specialized computer languages (such as an information retrieval language) can be formulated using a simple precedence grammar without too restrictive a language. This will eliminate the local ambiguity problem and will allow one to concentrate fully on the development of his specialized language.

Wirth and Weber (1966a, 1966b) have developed excellent parsing techniques for simple precedence grammars and have succeeded in separating the semantics so that the syntactic parsing controls when the semantics will be employed. Due to the simple precedence requirement, when a string is to be reduced, then there exists a unique production for this reduction and hence a unique semantic rule to be executed.

The work reported herein is mostly a consolidation of the work of Wirth and Weber and a translation of this work to the PL/1 programming language.

Introduction and Definition of Terms

The methods and definitions of Wirth and Weber (1966a) can be best illustrated through an example:

Suppose a language is desired which will calculate the sum of an arbitrary number of numbers. The syntax and associated semantics can be written as:

	Syntax		Semantics
p1	PROGRAM	::= BEGIN EQLISTA	Reinitialize parser
p2	EQLISTA	::= EQLIST	Null
p3	EQLIST	::= EQUATION	Null
p4		::= EQLIST ; EQUATION	Null
p5	EQUATION	::= IDENTIFIER = SUM	Print out identifier and value of sum
p6	SUM	::= (EXPRESSION)	Set the value of SUM to the value of EXPRESSION
p7	EXPRESSION	::= FACTOR	Null
p8	FACTOR	::= NUMBER	Null
p9		::= FACTOR + NUMBER	Set the value of FACTOR on the left to the sum of the value of FACTOR on the right and the value of NUMBER

Non-basic symbols are those which appear on the left (PROGRAM, EQLISTA, EQLIST, EQUATION, SUM, EXPRESSION, and FACTOR). Basic symbols are those which appear only on the right (BEGIN, ;, IDENTIFIER, =, (,), NUMBER, and +). Note that these are the language elements and the syntax determines how they may appear.

When the string S_jS_k appears to the parser, there are only three ways in which it may be parsed:

1. .....S_j S_k..... i.e., $S_j < S_k$
   reducible substring
2. ..... S_j S_k... i.e., $S_j > S_k$
   reducible substring
3. .....S_j S_k..... i.e., $S_j = S_k$
   reducible substring

These precedence relations may be formally defined by:

1. $S_j = S_k$ iff there is a rule $U ::= xS_jS_ky$;
2. $S_j < S_k$ iff there is a rule $U ::= xS_jU_iy$ and a rule $U_i ::=^* S_kz$;
3. $S_j > S_k$ iff [(there is a rule $U ::= xU_mS_ky$ and a rule $U_m ::=^* zS_j$) or (there is a rule $U ::= xU_mU_iy$ and a rule $U_m ::=^* zS_j$ and a rule $U_i ::=^* S_kw$)]. (Shaw, 1966)

Where

A::=B means A directly produces B (i.e., in only one step), and

A ::=* B means A indirectly produces B (i.e., in a finite number of steps).

Precedence relations for the example written in matrix form may be found on the next page. Suppose it is desired to parse:

BEGIN IDENTIFIER=(NUMBER + NUMBER).[1]

Start at the left and look to the right until a > relation is found or the end of the string. Now NUMBER>+, hence set k to 5 (i.e., the first NUMBER), indicating that the first greater than relation occurred at the fifth element

[1] As will be discussed later, '.' is the TERMINAL variable which is used to force the parsing until completion. Its value is set to zero, hence all symbols in the syntax are > than TERMINAL.

The precedence relations may be written in a matrix for the example as:

	PROGRAM	EQLISTA	EQLIST	EQUATION	SUM	EXPRESSION	FACTOR	BEGIN	IDENTIFIER	(	NUMBER	;	=	+	)
PROGRAM															
EQLISTA															
EQLIST												=			
EQUATION												>			
SUM												>			
EXPRESSION															=
FACTOR														=	>
BEGIN		=	<	<					<						
IDENTIFIER													=		
(						=	<				<				
NUMBER														>	>
;				=					<						
=					=					<					
+											=				
)												>			

To determine the relation between S_j and S_k, look in the row determined by S_j and the column determined by S_k. The blank entries in the matrix indicate that those symbols may never occur adjacently.

in the stack. Now scan to the left as long as = holds or to the beginning of the string, hence set j to 5. Now the reducible string is $S_j \ldots\ldots S_k$, i.e., 'NUMBER' and the corresponding production is p8. Replace the reducible string by the left side of p8 (i.e., FACTOR) and set k to point at this. The string becomes

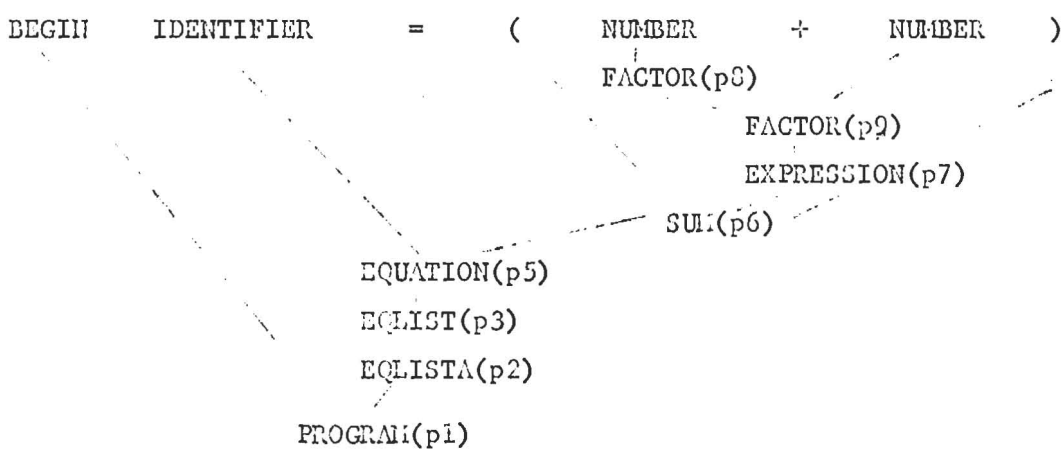
BEGIN IDENTIFIER=(FACTOR + NUMBER).

If > still holds between S_k and S_{k+1}, then the right limit is known and the scan to the left can be initiated, otherwise scan to the right until the > relation holds. At the end of this operation k will be 7 and j will be 5. The reducible string thus corresponds to p9 and the resultant string is:

BEGIN IDENTIFIER=(FACTOR).

The next resultant string is:	BEGIN IDENTIFIER=(EXPRESSION).
Then,	BEGIN IDENTIFIER=SUM.
Then,	BEGIN EQUATION.
Then,	BEGIN EQLIST.
Then,	BEGIN EQLISTA.
And, finally,	PROGRAM , at which point the parsing stops.

The complete parsing tree is:



Note that backtracking the value of the pointer k was not necessary (this was formally proven by Wirth and Weber, 1966a). Further if the language is defined as a left recursive language then reduction will take place as soon as possible within the scan, thereby reducing the length of the string which must be saved. In actual use the parsing is accomplished using stack techniques and a parallel stack for evaluating.

Also note that the precedence matrix is sparse, and thus Wirth and Weber (1966a) introduced precedence functions to economize on storage requirements. These functions are defined by:

1. $F(S_j) = G(S_k)$ iff $S_j = S_k$
2. $F(S_j) < G(S_k)$ iff $S_j < S_k$
3. $F(S_j) > G(S_k)$ iff $S_j > S_k$

where the operators as applied to the functions are the normal arithmetic operators. The use of this function notation actually destroys the blank entries in the precedence matrix and thus complicates error diagnostics.

The SARPSIS System

SARPSIS consists of four distinct operations:

1. ANALYZER
2. RECOGNIZER
3. PARSER
4. SEMANTIC INTERPRETER

These four operations will be illustrated using the preceding example.

ANALYZER

The purpose of the ANALYZER is to process the syntax and to generate the tables necessary for the parser and to punch these tables. The organization of the card deck to be read by the ANALYZER section is as follows:

VARIABLE	TYPE	DESCRIPTION
NUMBER	CHAR(12) VAR	That string in the syntax which will be recognized by the recognizer as a number.[2]
WORD	CHAR(12) VAR	That string in the syntax which will be recognized by the recognizer as a word.[2]
QUOTES	CHAR(12) VAR	That string not in the syntax which will be used to force the word recognition class.[2]

[2]The use of these five variables is explained in further detail in the Parser and Recognizer sections.

VARIABLE	TYPE	DESCRIPTION
TERMINAL	CHAR(12) VAR	That string which is not in the syntax which will be used to force the completion of the parsing.[2]
SEQUENCE	CHAR(12) VAR	That string in the syntax at which point the parsing may be re-initialized.[2]

Following these five variables is the option card which selects the desired output options of the Analyzer. Normally 'END' ending in column 79 is all that is necessary. Additional options may be requested by preceding the 'END' by the name of the options desired. The options are:

	Description
PRODUCTIONS	Productions printed in string form;
NUMPRD	Productions printed in number form;
SYMBOLS	The basic and non-basic symbols are printed with their unique number equivalent.
LEFT-RIGHT	The leftmost and rightmost symbols for all non-basic symbols are printed.
MATRIX	The precedence matrix is printed.
FUNCTIONS	The precedence functions are calculated and printed.
KEY-PRTB	The KEY and PRTB tables are computed and printed.
PUNCH	The data necessary for the parser is punched.

All but NUMPRD and LEFT-RIGHT are selected by default; these two are sometimes useful for debugging purposes.

The syntax cards in BNF follow the option card. The conventions used in these cards are:

1. All elements are less than 13 characters;
2. Blank is the separator between elements;
3. Each equation is limited to 6 elements and to one card;
4. The ::= is assumed between the first and second elements (note that it is never entered on the BNF cards);
5. If the first column is blank then the first element is the same as on the previous card;
6. A $ in column 1 indicates the end of the syntax.

The analyze procedure is divided into four steps: initialization; STEPA; STEPB; and STEPC. During the initialization step, the five variables and the option card are read and the output options are set.

In STEPA the syntax is read and the productions are printed out in a standard form. The productions are then scanned to determine the basic and non-basic symbols and these are assigned unique numbers (starting from 1). The productions (in string form) are then converted to numerical form using these unique numbers. The KEY and PRTB tables are then calculated using the numerical form. KEY[i] represents, for the i'th symbol the index in the production table PRTB, where those productions are listed whose right part string begins with the i'th symbol. For each production, the right part is listed without its leftmost symbol, followed by the negative of the production number and the left part symbol of the production. The end of the list of productions referenced by KEY[i] is marked with a 0 entry in PRTB. (Shaw, 1966).

In STEPB, the numerical form of the productions are scanned to determine the leftmost [L(U)] and the rightmost [R(U)] symbols of the non-basic symbols. These are formally defined by:

$L(U)=[S \mid \exists z(U::=*Sz)]$

$R(U)=[S \mid \exists z(U::=*zS)]$ (Wirth & Weber, 1966a)

L(U) is the set of symbols which appear at the left of any string which may be generated by applying any production starting with the symbol U as the initial string. Similarly for R(U).

In STEPC, the precedence matrix and the precedence functions are determined. The precedence matrix is determined by using the following alternate definitions of the precedence relations:

1. $S_j = S_k$ iff there is a rule $U::=S_jS_k$
2. $S_j < S_k$ iff there is a rule $U::= xS_jU_iy$ and $S_k \in L(U_i)$
3. $S_j > S_k$ iff [(there is a rule $U::= xU_mS_ky$ and $S_j \in R(U_m)$)
   or (there is a rule $U::= xU_iU_my$ and $S_j \in R(U_i)$ and
   $S_k \quad L(U_m)$)] (Wirth & Weber, 1966a, Shaw, 1966)

The precedence functions are then determined by an algorithm (Shaw, 1966; Wirth, 1965) which essentially permutes the rows and columns of the precedence matrix until the precedence relations are grouped.

The ANALYZER is an externally callable procedure in PL/1. It not only provides tables by punching but also returns these tables to the calling program via the parameter list.

The input cards, the printed output, and the punched output are illustrated in Appendix A. All possible diagnostic messages and their meanings are given in Appendix B; also included is a discussion about other types of undetected errors.

Recognizer

The function of the recognizer is to scan the input text for the next syntactical unit and to assign this unit the number which the analyzer had assigned to it. To accomplish this, the recognizer is divided into two major areas:

1. LOOK
2. ASSIGN

LOOK is a procedure which scans the input device and returns the next syntactical unit in string form. Currently it is written to read one card from the card reader and to only read another when the entire card is exhausted. Look assumes that there are two kinds of syntactical units; words and numbers. In all cases, a change in the class of characters or a blank terminates the scan. These two units are actually defined by Look through programming to conform to the following definitions:

```
NUMBER ::= DIGIT
       ::= NUMBER  DIGIT
       ::= +       DIGIT
       ::= -       DIGIT
       ::= NUMBER  .
WORD   ::= Special character (+ = & * : " etc.)
       ::= PHRASE
PHRASE ::= LETTER (A ... Z)
       ::= PHRASE  LETTER
```

e.g.,

(+2.0+ABC,-3.0.2) would be scanned by Look as:

```
(       WORD
+2.0    NUMBER
+       WORD
ABC     WORD
,       WORD
-3.0.2  NUMBER
)       WORD
```

Some improvement in the definitions may be needed later, but for the moment, this is satisfactory. Perhaps it would be better said that LOOK returns two kinds of strings; numbers and non-numbers (i.e., words).

ASSIGN is the procedure which calls Look and then assigns a number to the syntactical unit returned by Look. It accomplishes this task through two arrays provided by the ANALYZER; BASSYM and BASVAL. BASSYM contains all of the basic symbols in the syntax except those designated by the variables WORD and NUMBER (two of the five variables which were input to the analyzer). In addition, BASSYM contains the symbol designated by the variable TERMINAL (another of the variables input to the Analyzer). BASVAL contains all of the numbers associated with the entries in BASSYM.

If LOOK signifies that the symbol is a number, then the value corresponding to NUMBER is returned. However if LOOK signifies that the symbol is a WORD, then ASSIGN performs a simple table lookup to determine if it is a basic symbol; if so, then the corresponding number from BASVAL is returned else the number corresponding to WORD is returned.

These basic symbols correspond to reserved words in programming languages. The objective here was to provide a method of using reserved words so that they would not be recognized as reserved words in certain contexts. The motivation is from the information retrieval viewpoint. Suppose an author existed whose keyword was AUTHOR -- How would one indicate an author search for JOHN Q. AUTHOR?

The solution to this problem in ASSIGN was to provide an override feature upon the BASSYM table search. This is accomplished by making ASSIGN

recursive and by using the variable QUOTES (another one of the five variables input to the Analyzer). As indicated, QUOTES is a symbol not in the syntax of the language defined. Within ASSIGN, QUOTES controls a Boolean switch If the switch is true then no table search is performed. When QUOTES occurs within an input stream, then ASSIGN logically complements this Boolean switch and calls itself recursively to return the next syntactical unit in the input stream. The principal requirement upon QUOTES is that it must not be recognizable as a number as defined by LOOK.

PARSER

The PARSER is the main program and, as such, decides which of its auxiliary functions to call. Upon invocation the PARSER first reads a data card to control the dimensioning of the arrays; these are the dimensions in effect when and if the ANALYZER is called. These dimensions are specified on the first card of the data deck in the following order:

M number of symbols in the syntax

N number of equations in the syntax

MI number of basic symbols in the syntax

These values must be slightly larger than the values calculated by the ANALYZER for some temporary storage during the Analyzer process (see Appendix B).

The following procedures are auxiliary functions of the PARSER (i.e., called by the Parser):

1. SETUP
2. ERRORREC
3. STACKOK
4. ASSIGN (i.e., the Recognizer)
5. SEMANT (i.e., the Semantic Interpreter)

After dimensioning the arrays, the PARSER calls the SETUP procedure to initialize those arrays needed in order to parse an input string.

The following procedures are auxiliary functions of SETUP:

1. ANALYZE (i.e., the Analyzer)
2. TABREAD
3. TABPRINT

SETUP first reads a data card to determine whether to call the ANALYZER (i.e,, BNF syntax cards follow) or to call the TABREAD procedure (i.e., the data cards punched by a previous invocation of the ANALYZER follow). The option is signified by 'ANALYZE' (call Analyzer) or 'TABLE' (call TABREAD) on this data card. In any event SETUP prints its own name and the value on this first card. After the ANALYZER or the TABREAD procedure has returned control to SETUP, SETUP reads a data card with three variables on it:

Variable	Type	Description
ERSCAN	CHAR(12)	That symbol in the syntax to which the ERRORREC procedure will be allowed to scan forward to if a syntax error develops. In normal cases, all input between two of these symbols will be ignored if a syntax error is detected. (in the example ";")
SYMA	CHAR(12)	That symbol in the syntax which is normally expected to reside in the first element of the parsing stack (in the example BEGIN).
SYMB	CHAR(12)	That symbol in the syntax which is normally expected to reside in the second element of the parsing track (in the example EQLIST).

After reading these three variables, SETUP calls the TABPRINT procedure and then returns to the PARSER.

The TABREAD procedure is designed to read the data cards punched by the ANALYZER. It prints the name of all variables read. A sample printout resulting from SETUP calling TABREAD for the example is given in Appendix C.

The TABPRINT procedure prints the contents of all the arrays read in by TABREAD. A sample printout for the example is given in Appendix D.

After the PARSER has called SETUP, then the system is ready for input strings to be parsed in the defined language. The PARSER uses its auxiliary functions to scan the input string from left to right and to parse the resultant string. Its operation is explained by the flow chart of Appendix E.

As indicated in the PARSER flow chart, STACKOK is called before any string is reduced. STACKOK checks to see if the leftmost symbol of the string to be reduced can occur adjacent to the symbol to the left of it in the symbol stack (S). If they cannot occur adjacent, then a syntax error has occurred and ERRORREC is called.

ERRORREC rescans the current input string to check for syntax errors. As discussed earlier, the precedence functions complicate syntax error checking, hence a compact form of the precedence matrix (HM) is used for ERRORREC and STACKOK. ERRORREC uses the variables ERSCAN, SYMA, and SYMB to reset the parsing (S) and value stack (VS) to their values before the error and to advance the input scanner past the syntax error. Note that since all the information used by these two routines is either prepared by ANALYZE (or equivalently read in by TABREAD) the error recovery and syntax checking is governed only by the syntax and the variables input.

If the current string to be reduced is not in error then the semantic interpretation procedure (SEMANT) is called to apply the interpretation rule corresponding to the equation number determined by the Parser.

SEMANTIC INTERPRETER

The semantic interpreter is an external procedure which must be programmed in PL/1 for each different syntax. The procedure name is SEMANT and the parameters are:

Name	Type	Function
N	FIXED BINARY	Formula number
VS	ARRAY(0:50) CHAR(400) VARYING	Value stack
J	FIXED BINARY	Left-hand stack pointer
K	FIXED BINARY	Right-hand stack pointer
EQ	ARRAY(U) CHAR(12)	EQ(I) is the left-hand side for syntax rule I
ANS	FIXED BINARY	For use in SEMANT, set to zero initially
SWITCH	BIT(1)	For use in SEMANT, set to False initially
U	FIXED BINARY	Upper bound for EQ

SEMANT is called only when a string is to be reduced. When it is called, N is set to the equation number corresponding to the syntax rule. J is set to the beginning of the string to be reduced and K is set to the end of this string. The function of SEMANT is to update the value stack corresponding to J, K, N; the value stack is the only place where the results of semantic interpretation have an effect. Clearly not all situations will be able to utilize this value stack exclusively, however it may still be used to point to other data areas which can easily be added.

The SEMANT procedure for our example is given in Appendix F. The output from this run is Appendix G.

Conclusion

SARPSIS provides a convenient way of defining and implementing specialized computer languages. It analyzes the syntax, parses any input string and calls a semantic interpretation program. Error messages are automatic and depend only on the syntax. The only programming required is the Semantic Interpretation, which may indeed be a large task. However, it does allow one to concentrate fully on the semantics of his language and frees him from analyzing, scanning, and parsing problems.

A listing of ANALYZE and PARSER forms Appendices H and I.

J. E. George

References

Wirth, Niklaus and Helmut Weber. "EULER: A Generalization of ALGOL, and its Formal Definition: Part I", Communications of the ACM, Vol. 9, No. 1, (January 1966), pp. 13-25.

Wirth, Niklaus and Helmut Weber. "EULER: A Generalization of ALGOL, and its Formal Definition: Part II", Communications of the ACM, Vol. 9, No. 2, (February 1966), pp. 89-99.

Reynolds, John C. "COGENT Programming Manual", Argonne National Laboratory Report No. ANL-7022, March 1965.

Shaw, Alan C. "Lecture Notes on a Course in Systems Programming", Computer Science Department, Stanford University, Techical Report No. 52 (December 1966).

Wirth, Niklaus. "Find Precedence Functions", Algorithm 265, Communications of the ACM, Vol. 8, No. 10 (October 1965), pp. 604-605.

APPENDIX A

Input and output for the example

INPUT DECK

```
'NUMBER'   'IDENTIFIER'   ''''    '.'    'PROGRAM'   (i.e., the five variables)
'LEFT-RIGHT'                                   'END'(options card)
PROGRAM        BEGIN       EQLISTA
EQLISTA        EQLIST
EQLIST         EQUATION
               EQLIST      ;            EQUATION
EQUATION       IDENTIFIER  =            SUM
SUM            (           EXPRESSION   )
EXPRESSION     FACTOR
FACTOR         NUMBER
               FACTOR      +            NUMBER
$
```

```
SETUP
ANALYZE
START TIME=                172008490
OPTIONS SELECTED
PRODUCTIONS
SYMBOLS
LEFT-RIGHT
MATRIX
FUNCTIONS
KEY-PRTB
PUNCH
```

PRODUCTIONS

1	PROGRAM	::=	BEGIN	EQLISTA	
2	EQLISTA	::=	EQLIST		
3	EQLIST	::=	EQUATION		
4		::=	EQLIST	;	EQUATION
5	EQUATION	::=	IDENTIFIER	=	SUM
6	SUM	::=	(	EXPRESSION	)
7	EXPRESSION	::=	FACTOR		
8	FACTOR	::=	NUMBER		
9		::=	FACTOR	+	NUMBER

NON-BASIC SYMBOLS

1	PROGRAM	2	EQLISTA
3	EQLIST	4	EQUATION
5	SUM		
6	EXPRESSION	7	FACTOR

BASIC SYMBOLS

8	BEGIN	9	IDENTIFIER
10	(	11	NUMBER
12	;		
13	=	14	+
15	)		

KEY

0	0	0	1	8	10	10	11	18	22
27	32	34	34	34	34	34			

PRT3

0	-2	2	12	4	-4	3	0	-3	3
0	-7	6	14	11	-9	7	0	2	-1
1	0	13	5	-5	4	0	6	15	-6
5	0	-8	7	0					

```
RIGHT SYMBOL SETS
PROGRAM      ::=  EQLISTA      EQLIST      EQUATION      SUM          )
EQLISTA      ::=  EQLIST       EQUATION    SUM           )
EQLIST       ::=  EQUATION     SUM         )
EQUATION     ::=  SUM          )
SUM          ::=  )
EXPRESSION   ::=  FACTOR       NUMBER
FACTOR       ::=  NUMBER
```

```
LEFT SYMBOL SETS
PROGRAM      ::=  BEGIN
EQLISTA      ::=  EQLIST        EQUATION       IDENTIFIER
EQLIST       ::=  EQLIST        EQUATION       IDENTIFIER
EQUATION     ::=  IDENTIFIER
SUM          ::=  (
EXPRESSION   ::=  FACTOR        NUMBER
FACTOR       ::=  FACTOR        NUMBER
```

```
PRECEDENCE FUNCTIONS F AND G
  1 PROGRAM            1      1
  2 EQLISTA            1      1
  3 EQLIST             1      2
  4 EQUATION           2      2
  5 SUM                2      1
  6 EXPRESSION         1      1
  7 FACTOR             2      2
  8 BEGIN              1      1
  9 IDENTIFIER         1      3
 10 (                  1      2
 11 NUMBER             3      2
 12 ;                  2      1
 13 =                  1      1
 14 +                  2      2
 15 )                  2      1
FINISH TIME=              172010950
```

```
PRECEDENCE MATRIX
                 1
   1             .        .
   2             .        .
   3             .   =    .
   4             .   >    .
   5             .   >    .
   6             .      =.
   7             .     =>.
   8     =<<     <.       .
   9             .    =   .
  10        =<   . <      .
  11             .     >>.
  12       =     <.       .
  13        =     .<      .
  14             . =      .
  15             .  >     .
NO PRECEDENCE VIOLATIONS OCCURRED
```

PUNCHED OUTPUT

```
$ EBD
"TABLE                                                                                                                                  "
"MNMM                                                                                                                                   "
  15     9     7
"XNUM-XWORD-XSEQ                                                                                                                        "
  11     9     1
"QUOTES                                                                                                                                 "
""                                                                                                                                      "
"EQUATIONS                                                                                                                              "
"PROGRAM          " "EQLISTA        " "EQLIST         " "EQLIST         " "EQUATION       "
"SUM              " "EXPRESSION     " "FACTOR         " "FACTOR         "
"BASSYM                                                                                                                                 "
"BEGIN            " "(              " ";              " "=              " "+              "
")                " ","             "
"BASVAL                                                                                                                                 "
   8    10    12    13    14    15    16
"SYMBOLS                                                                                                                                "
"                 " "PROGRAM        " "EQLISTA        " "EQLIST         " "EQUATION       "
"SUM              " "EXPRESSION     " "FACTOR         " "BEGIN          " "IDENTIFIER     "
"(                " "NUMBER         " ";              " "=              " "+              "
")                "
"KEY                                                                                                                                    "
   0     0     0     1     8    10    10    11    18    22    27    32    34    34    34
  34    34
"PRTB                                                                                                                                   "
   0    -2     2    12     4    -4     3     0    -3     3     0    -7     6    14    11
  -9     7     0     2    -1     1     0    13     5    -5     4     0     6    15    -6
   5     0    -8     7     0
"HM                                                                                                                                     "
   1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   2    12     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   2    12     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   2    12     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   2    15     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   3    15    14     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   5     2     3     4     9     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0
   2    13     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

```
    0
    4    6    7   11    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0
    3   15   14    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0
    3    4    9    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0
    3    5   10    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0
    2   11    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0
    2   12    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0
"F                                                                        "
    0    1    1    1    2    2    1    2    1    1    1    3    2    1    2
    2    0
"G                                                                        "
    0    1    1    2    2    1    1    2    1    3    2    2    1    1    2
    1    0
"END                                                                      "
```

APPENDIX B

Analyzer Error Messages

Message	Solution
QUOTES AND/OR TERMINAL MAY NOT APPEAR IN SYNTAX PRECEDENCE VIOLATION	Rewrite the syntax so that the value assigned to the variable QUOTES and/or terminal does not appear in the syntax
PRECEDENCE VIOLATION OCCURRED	Rewrite syntax
HINTS REGARDING ERRORS	Tells between what two symbols and equation the precedence violation occurred
MM MATRIX FULL AT ROW X	Increase dimension of MM in Parser and Analyzer
NO PRECEDENCE FUNCTIONS EXIST	Rewrite syntax (reference 1)

Some rules to follow when writing the syntax are:

1. Use left recursive definitions only;
2. If a phrase class is recursive, then introduce one more level in the syntax, e.g.

```
phrase::=phrase                          phrase::=word
phrase::=word            instead of      phrase::=word phrase
      ::=phrase word
```

The program which calls ANALYZE controls all the dimensioning of arrays in ANALYZE; these in turn are estimated by an input card (see parser section). Should these dimensions be too small fatal trouble can result. This trouble is characterized by:

1. Looping;
2. Negative entries in KEY;
3. Blank symbols in the precedence function table.

A good rule of thumb in the beginning is to use the number of equations + 10 for the variables M N MM in the parser. For data runs, the values punched by the analyze routine (MMMM) + 3 may be used.

Appendix C

Output from SETUP and TABREAD

```
SETUP
TABLE
TABREAD
MNMM
XNUM-XWORD-XSEQ
QUOTES
EQUATIONS
BASSYM
BASVAL
SYMBOLS
KEY
PRTB
HM
F
G
END
```

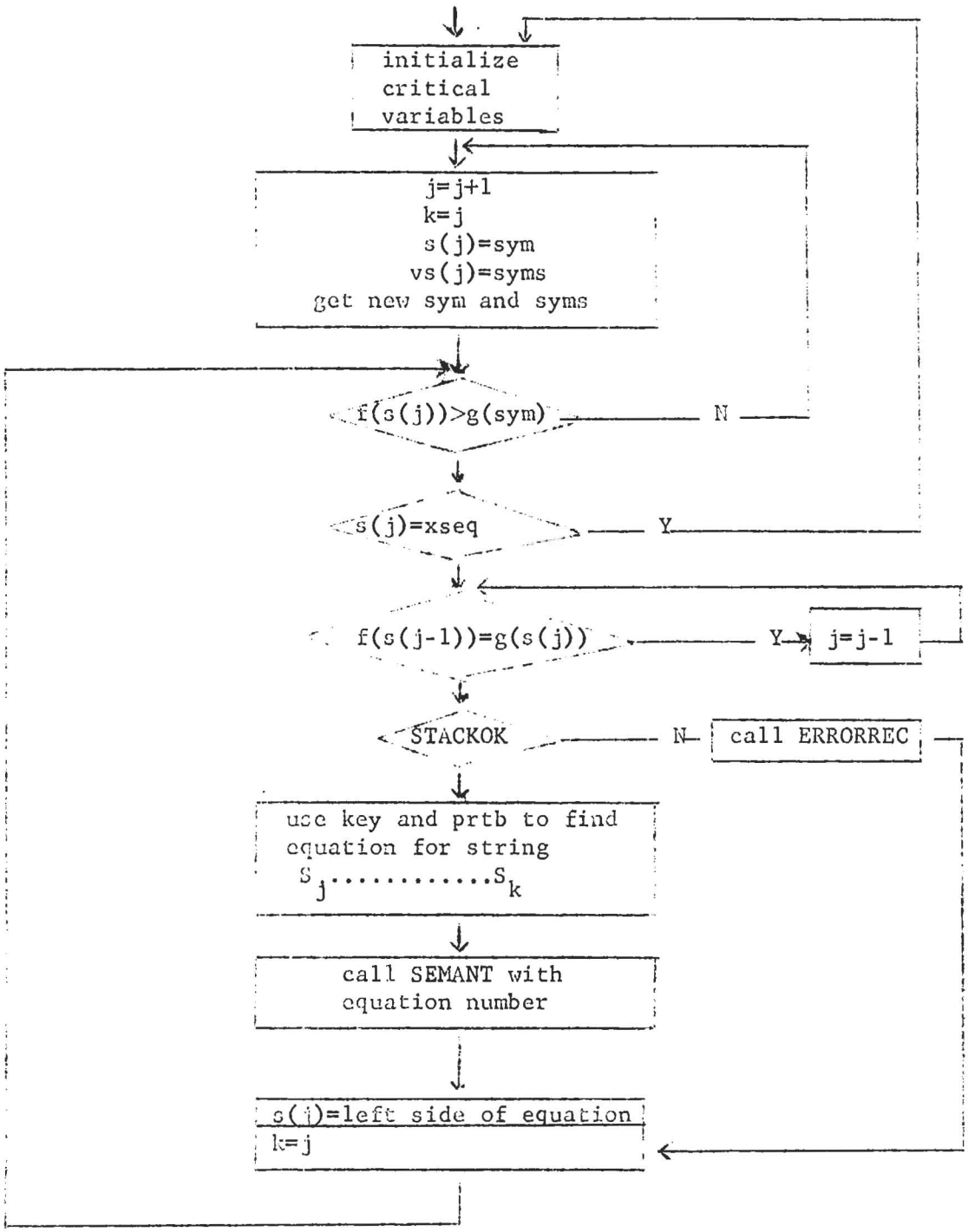
Appendix D Sample Output from TABPRINT

```
TABLE ENTRIES        M=     15        N=       9        MM=       7
ERSCAN=; SYMA=BEGIN SYMB=EQLIST
   XWORD=    9    XNUM=   11    XSEQ=    1    QUOTES="
F
      0       1       1       1       2       2       1       2       1       1
      1       3       2       1       2       2       0
G
      0       1       1       2       2       1       1       2       1       3
      2       2       1       1       2       1       0
KEY
      0       0       0       1       8      10      10      11      18      22
     27      32      34      34      34      34      34
PRTB
      0      -2       2      12       4      -4       3       0      -3       3
      0      -7       6      14      11      -9       7       0       2      -1
      1       0      13       5      -5       4       0       6      15      -6
      5       0      -8       7       0
BASVAL
      8      10      12      13      14      15      16
BASSYM
     BEGIN                  (                      ;                      =
     +                      )                      .
EQUATIONS
     PROGRAM                EQLISTA                EQLIST                 EQLIST
     EQUATION               SUM                    EXPRESSION             FACTOR
     FACTOR
SYT
                            PROGRAM                EQLISTA                EQLIST
     EQUATION               SUM                    EXPRESSION             FACTOR
     BEGIN                  IDENTIFIER             (                      NUMBER
     ;                      =                      +                      )
HM
     1       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
     1       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
     2      12       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
     2      12       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
     2      12       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
     2      15       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
     3      15      14       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0
```

5	2	3	4	9	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
2	13	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
4	6	7	11	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
3	15	14	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
3	4	9	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
3	5	10	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
2	11	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	
2	12	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	

Appendix E

Parser Flow Chart



```
PROCEDURE(N,VS,J,K,EQ,ANS,SWITCH,U);

SEMANT:     PROCEDURE(N,VS,J,K,EQ,ANS,SWITCH,U);
            DCL U FIXED BINARY;  /* UPPERBOUND FOR EQ  */
            DCL N FIXED BINARY;  /* FORMULA NUMBER */
            DCL VS(0:50) CHAR(400) VAR;    /*  VALUE STACK  */
            DCL J FIXED BINARY;  /* LEFT HAND STACK POINTER */
            DCL K FIXED BINARY;  /* RIGHT HAND STACK POINTER */
            DCL EQ(U) CHAR(12); /*EQ(N)=LEFT HAND SIDE CHAR STR FOR*/
                              /*   FORMULA N  */
            DCL ANS FIXED BINARY; /*SET TO 0 USED ONLY IN SEMANTICS*/
            DCL SWITCH BIT(1);/* SET TO FALSE USED ONLY BY SEMANTICS*/
           PUT EDIT('FORMULA  ',N,'<',EQ(N),'>')(SKIP,A,F(4),X(2),3 A);
            IF N=5 THEN PUT EDIT(VS(J),'=',VS(K))(SKIP,3 A);
            IF N=6 THEN VS(J)=VS(J+1);
            IF N=9 THEN VS(J)=VS(J)+VS(K);
                END SEMANT;
```

APPENDIX F

SEMANT PROCEDURE

```
   BEGIN      A=(1 + 2 + 3 );
FORMULA      8  <FACTOR       >
FORMULA      9  <FACTOR       >
FORMULA      9  <FACTOR       >
FORMULA      7  <EXPRESSION   >
FORMULA      6  <SUM          >
FORMULA      5  <EQUATION     >
A=         6
FORMULA      3  <EQLIST       >
```

APPENDIX G.1

SEMANT OUTPUT

```
                                        A=(1 + 2  2 );
FORMULA      8  <FACTOR        >
FORMULA      9  <FACTOR        >
SYNTAX ERROR--- IF MORE INPUT REQUESTED BEFORE END
OF SYNTAX ANALYSIS THEN PLEASE ENTER (;)
NUMBER        MAY NOT BE FOLLOWED BY NUMBER
END OF SYNTAX ERROR ANALYSIS
```

```
                                        B=(1 + 2) .
FORMULA      8   <FACTOR        >
FORMULA      9   <FACTOR        >
FORMULA      7   <EXPRESSION    >
FORMULA      6   <SUM           >
FORMULA      5   <EQUATION      >
B=           3
FORMULA      4   <EQLIST        >
FORMULA      2   <EQLISTA       >
FORMULA      1   <PROGRAM       >
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,

ANALYZE:     PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,
                 BASVAL,BASSYM,EQ,XSEQ,ERRORFLAG,HM,SYT);
             /* CREATES PARSING TABLES FROM SYNTAX IN BNF */
             /* INPUT SEQUENCE    */
             /* CARD 1  NAME USED IN SYNTAX FOR NUMBER  */
             /* CARD 1  NAME USED IN SYNTAX FOR WORD  */
             /* CARD 1  NAME USED IN SYNTAX FOR QUOTES */
             /* CARD 1  NAME USED IN SYNTAX FOR TERMINAL  */
             /* CARD 1  NAME USED IN SYNTAX FOR SEQUENCE  */
             /* CARD 2  OPTIONS    */
                 /*        PRODUCTIONS,NUMPRD,SYMBOLS,LEFT-RIGHT,*/
             /*   MATRIX,FUNCTIONS,KEY-PRTB,PUNCH,END    */
             /*  NORMALLY  'END' STARTING IN COL 75    */
         /*  SYNTAX IN BNF    */
             /*   $  IN COL 1   DENOTES END OF SYNTAX  */
         DCL X CHAR(12) VAR,IN CHAR(100) VAR,DATAIN(80) CHAR(1);
         DCL (I,J,K) FIXED BINARY;
         DCL M FIXED BINARY;                    /* NUMBER OF SYMBOLS  */
         DCL MM FIXED BINARY;                   /* NO NON-BASIC SYMBOLS */
         DCL N FIXED BINARY;                   /* NUMBER OF PRODUCTIONS*/
         DCL ERRORFLAG BIT(1);
         DCL CHANGE BIT(1);
         DCL HM(M,0:30) FIXED BINARY;  /* MATRIX FOR DIAGNOSTICS  */
         DCL SYT(0:M)   CHAR(12);    /* SYMBOL TABLE                  */
         DCL PRD(0:N  ,0:5) FIXED BINARY; /* PRD IN NUMBER FORM*/
         DCL H(0:M,0:M) CHAR(1);     /*  PRECEDENCE MATRIX     */
         DCL (F(0:M),G(0:M)) FIXED BINARY; /*PRECEDENCE FUNCTIONS */
         DCL L(0:MM,0:M) BIT(1),R(0:MM,0:M) BIT(1);
             /*  L(I,J) TRUE MEANS THAT SY-J OCCURS IN THE */
             /* LEFT SYMBOL SET OF SY-I.R(I,J) MEANS THAT  */
             /*  SY-J IS IN RIGHT OF SY-I*/
         DCL (A,B,NN) FIXED BINARY;
         DCL XX(10) CHAR(15) VAR;
         DCL CHOICE(10) BIT(1);  /*  COMPUTATION AND OR PRINT OPTIONS*/
         DCL (KEY(0:M  ),PRTB(0:5*N)) FIXED BINARY;
         DCL COM CHAR(76);
         DCL (NUMBER,QUOTES,WORD,TERMINAL,SEQUENCE) CHAR(12);
         DCL (XNUM,XWORD,XSEQ)  FIXED BINARY;
         DCL (FMIN,GMIN,K1) FIXED BINARY;
         DCL BASVAL(M  ) FIXED BINARY;
         DCL BASSYM(M  ) CHAR(12);
         DCL EQ(N  ) CHAR(12);
         DCL (XM,XN,XMM) FIXED BINARY;
INTPNCH:     PROCEDURE(S,L,U,LL,UU,N,W);
             DCL L  FIXED BINARY;   /*  LOWER BOUND OF S  */
             DCL U FIXED BINARY;  /*  UPPER BOUND OF S  */
             DCL LL FIXED BINARY;/* FIRST ITEM TO BE PUNCHED  */
             DCL UU FIXED BINARY;  /*  LAST ITEM TO BE PUNCHED  */
             DCL N FIXED BINARY;  /*  NUMBER OF ITEMS PER CARD   */
             DCL W FIXED BINARY; /*  NUMBER OF CHARACTERS PER ITEM  */
             DCL S(L:U) FIXED BINARY;   /*  ARRAY TO BE PUNCHED   */
             DCL (I,J,X) FIXED BINARY;
             IF N*(W+1)>=80 THEN PUT EDIT('FORMAT ERROR IN INTEGER
                 PUNCH','N=',N,'W=',W)(PAGE,A,X(2),A,F(8),X(2),A,X(2),
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRT8,QUOTES,XNUM,XWORD,

                F(3));
            ELSE DO;
                DO I=LL TO UU BY N;
                    IF I+N-1>UU THEN X=UU; ELSE X=I+N-1;
                    PUT FILE (SYSPNCH) EDIT((S(J) DO J=I TO X     ))
                       ((N)(X(1),F(W)));
                        J=79-(X-I+1)*(W+1);
                    PUT FILE (SYSPNCH) EDIT(' ')(X(J),A);
                    END;
                END;
            RETURN;
            END INTPNCH;
STRPNCH:    PROCEDURE(S,L,U,LL,UU,N,W);
            DCL L  FIXED BINARY;    /*  LOWER BOUND OF S   */
            DCL U FIXED BINARY;  /*  UPPER BOUND OF S   */
            DCL LL FIXED BINARY;/* FIRST ITEM TO BE PUNCHED   */
            DCL UU FIXED BINARY;  /*  LAST ITEM TO BE PUNCHED   */
            DCL N FIXED BINARY;  /*  NUMBER OF ITEMS PER CARD    */
            DCL W FIXED BINARY; /*  NUMBER OF CHARACTERS PER ITEM   */
            DCL S(L:U) CHAR(W);  /* STRING TO BE PUNCHED   */
            DCL (I,J,X) FIXED BINARY;
            IF N*(W+3)>=80 THEN PUT EDIT('FORMAT ERROR IN STRING
                PUNCH','N=',N,'W=',W)(PAGE,A,X(2),A,F(8),X(2),A,X(2),
                    F(3));
            ELSE DO;
                DO I=LL TO UU BY N;
                    IF I+N-1 > UU THEN X=UU; ELSE X=I+N-1;
                    DO J=I TO X;
                        PUT FILE (SYSPNCH) EDIT('''',S(J),'''')(X(1),3
                                        A);
                        END;
                    J=79-(X-I+1)*(W+3);
                    PUT FILE (SYSPNCH) EDIT(' ')(X(J),A);
                END;
                END;
            RETURN;
                END STRPNCH;
COMPNCH:    PROCEDURE(S);
            DCL S CHAR(76);
            DCL SS(0:1) CHAR(76);
            DCL (I,J,K) FIXED BINARY;
            I=0; J=1; K=76; SS(0)=S; SS(1)=S;
            CALL STRPNCH(SS,I,J,I,I,J,K);
            RETURN;
            END COMPNCH;
        /*   SET UP OUTPUT CONTROLS     */
            PUT LIST('START TIME=',TIME) SKIP;
        XN=N;    XM=M;    XMM=MM;
        XX(1)='PRODUCTIONS';
        XX(2)='NUMPRD';
        XX(3)='SYMBOLS';
        XX(4)='LEFT-RIGHT';
        XX(5)='MATRIX';
        XX(6)='FUNCTIONS';
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,

        XX(7)='KEY-PRTB';
        XX(8)='PUNCH';
        XX(9)=' ';
        XX(10)=' ';
        GET LIST(NUMBER,WORD,QUOTES,TERMINAL,SEQUENCE);
        DO I=1 TO 10; CHOICE(I)='0'B;  END;
        IN=' ';
        DO I=1,3,5,6,7,8;  CHOICE(I)='1'B;  END;
        DO WHILE (IN¬='END');
            GET LIST(IN);
            DO I=1 TO 10;  IF XX(I)=IN THEN CHOICE(I)='1'B; END;
            END;
        PUT LIST('OPTIONS SELECTED') SKIP;
        DO I=1 TO 10;  IF CHOICE(I) THEN PUT LIST(XX(I)) SKIP; END;
              SYT(0)=' ';  ERRORFLAG='0'B;  K=0;
        DO I=0 TO N  ;
            DO J=0 TO 5;
                PRD(I,J)=0;   END;
            END;
STEPA:  BEGIN;
        DCL P(0:N  ,0:5) CHAR(12);  /* PRODUCTIONS IN STRING FORM   */
        DCL (U,V) FIXED BINARY;
        /*  READ THE PRODUCTIONS TIL $ IS READ                       */
        DATAIN(1)=' ';              N=0;
READA:  DO WHILE (DATAIN(1)¬='$');
            GET EDIT((DATAIN(I) DO I=1 TO 80))(80 (A(1)));
            IF DATAIN(1)='$' THEN GO TO READA;
                N=N+1;   K=0;
            IF DATAIN(1)=' ' THEN DO;
                P(N,0)=' ';  K=1;   END;
            IN='';
            DO I=1 TO 80;     IN=IN||DATAIN(I);  END;  IN=IN||' ';
            I=LENGTH(IN);
            DO WHILE (K<=5 & I>0);
                X='';
                DO WHILE (SUBSTR(IN,1,1)¬=' ');
                    X=X||SUBSTR(IN,1,1);   IN=SUBSTR(IN,2); I=I-1; END;
                IF X¬='' THEN DO;
                    P(N,K)=X;
                    K=K+1;          END;
                DO WHILE (SUBSTR(IN,1,1)=' '&I>0); IN=SUBSTR(IN,2);
                    I=I-1;          END;
            END;
            DO I=K TO 5;   P(N,I)=' ';  END;
        END READA;
        /*    OUTPUT PRODUCTIONS                                     */
        IF CHOICE(1) THEN    AA: DO;
        PUT EDIT ('PRODUCTIONS',' ')(PAGE,A,SKIP,A);
OUT:    DO I=1 TO N;
            PUT EDIT(I,P(I,0),'::=',(P(I,K) DO K=1 TO 5))
                (SKIP,F(4),X(4),A,X(2),A,X(2),5 A) ;
            END OUT;
        END AA;
        /*  IDENTIFY AND LIST THE NON-BASIC AND BASIC SYMBOLS        */
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,

        M=0;
BASIC:  DO K=0 TO 5;
            DO I=1 TO N;
                X=P(I,K);
                DO J=0 TO M;  IF X=SYT(J) THEN GO TO FF; END;
                M=M+1;  J=M;  SYT(M)=X;
FF:             PRD(I,K)=J;
                END;
            IF K=0 THEN MM=M;
            END BASIC;
        DO I=1 TO N;
            IF PRD(I,0)=0 THEN DO;
                    PRD(I,0)=PRD(I-1,0);  P(I,0)=P(I-1,0); END;  END;
        IF CHOICE(3) THEN    AB: DO;
        PUT EDIT('NON-BASIC SYMBOLS')(PAGE,A);
        DO I=1 TO MM BY 5;
            IF I+4<MM THEN J=I+4; ELSE J=MM;
            PUT EDIT((K,SYT(K) DO K=I TO J  ))(SKIP,5(F(4),X(2),A,
                X(6)));
            END;
        PUT EDIT('BASIC SYMBOLS')(SKIP(2),A);
        DO I=MM+1 TO M BY 5;
            IF I+4<M  THEN J=I+4; ELSE J=M ;
            PUT EDIT((K,SYT(K) DO K=I TO J  ))(SKIP,5(F(4),X(2),A,
                X(6)));
            END;
        END AB;
        IF CHOICE(2) THEN  AC: DO;
        PUT EDIT('PRD')(PAGE,A);
        DO I=1 TO N;
            PUT EDIT(I,PRD(I,0),':::=',(PRD(I,J) DO J=1 TO 5))
                (SKIP,F(4),X(4),F(4),A,5(X(4),F(4)));
            END;
        END AC;
        /* PUNCH M,N,MM,EQ,BASSYM,BASVAL,NUMBER.WORD.QUOTES,XSEQ  */
        IF CHOICE(8) THEN   AP:  DO;
            COM='TABLE';  CALL COMPNCH(COM);
            COM='MNMM';
            CALL COMPNCH(COM);
            F(0)=M; F(1)=N; F(2)=MM;  I=0; J=XM ; K=3; A=4; B=2;
            CALL INTPNCH(F,I,J,I,B,K,A);
            DO I=1 TO M;
                IF SYT(I)=NUMBER THEN XNUM=I;
                IF SYT(I)=WORD THEN XWORD=I;
                IF SYT(I)=QUOTES | SYT(I)=TERMINAL THEN DO;
                    PUT EDIT('QUOTES AND/OR TERMINAL MAY NOT APPEAR IN'
                        ,' SYNTAX   PRECEDENCE VIOLATION')(SKIP,2 A);
                ERRORFLAG='1'B;  GO TO FIN;  END;
                IF SYT(I)=SEQUENCE THEN XSEQ=I;
                END;
            COM='XNUM-XWORD-XSEQ';
            CALL COMPNCH(COM);
            F(0)=XNUM; F(1)=XWORD; F(2)=XSEQ; K=2; I=0;
            CALL INTPNCH(F,I,J,I,K,A,A);
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,

        COM='QUOTES';
        CALL COMPNCH(COM);
      COM=QUOTES; CALL COMPNCH(COM);
        COM='EQUATIONS';   CALL COMPNCH(COM);
        DO I=1 TO N;  EQ(I)=P(I,0);  END;
        I=1;  J=XN ;  K=N;  A=5; B=12;
        CALL STRPNCH(EQ,I,J,I,K,A,B);
        K=1;
        DO I=MM+1 TO M;
        IF ¬(SYT(I)=NUMBER | SYT(I)=WORD | SYT(I)=QUOTES) THEN DO;
            BASSYM(K)=SYT(I); BASVAL(K)=I; K=K+1; END;
            END;
        COM='BASSYM';  CALL COMPNCH(COM);
        BASSYM(K)=TERMINAL;  BASVAL(K)=M+1;
        I=1;  J=XM;
        CALL STRPNCH(BASSYM,I,J,I,K,A,B);
        COM='BASVAL';
        CALL COMPNCH(COM);
        A=4; B=15;        CALL INTPNCH(BASVAL,I,J,I,K,B,A);
        A=5;  B=12;  I=0;    COM='SYMBOLS'; CALL COMPNCH(COM);
        CALL STRPNCH(SYT,I,XM,I,M,A,B);
        END AP;
    /*  COMPUTE AND OUTPUT KEY AND PRTB TABLES   */
    IF CHOICE(7) THEN  AE:  DO;
        K=0; V=0;  KEY(0)=0;  PRTB(0)=0;
        DO I=1 TO M+1;
            IF V¬=0 THEN KEY(I-1)=V;
            V=0;
            IF PRTB(K)¬=0 THEN K=K+1;
            PRTB(K)=0;  KEY(I)=K;
            DO J=1 TO N;
                IF PRD(J,1)=I THEN DO;
                        IF V=0 THEN V=K+1;
                    DO U=2 TO 5;
                        IF PRD(J,U)¬=0 THEN DO;
                        K=K+1;  PRTB(K)=PRD(J,U);          END;
                        END;
                K=K+1;  PRTB(K)=-J;  K=K+1; PRTB(K)=PRD(J,0);
                END;
            END;
        END;
        PUT EDIT('KEY')(PAGE,A);
        PUT EDIT((KEY(I) DO I=0 TO M+1))(SKIP,10 F(4));
        PUT EDIT('PRTB')(PAGE,A);
        PUT EDIT((PRTB(I) DO I=0 TO K))(SKIP,10 F(4));
        END AE;
    /*   PUNCH KEY AND PRTB     */
    IF CHOICE(8) THEN   APP:  DO;
        COM='KEY';
        CALL COMPNCH(COM);
        I=0;  J=XM ;  A=15;  B=4;        U=M+1;
        CALL INTPNCH(KEY,I,J,I,U,A,B);
        COM='PRTB';   CALL COMPNCH(COM);
        J=5*XN;  CALL INTPNCH(PRTB,I,J,I,K,A,B);
```

```
            END APP;
        END STEPA;
STEPB:  BEGIN;
        DCL (U,V) FIXED  BINARY;
        DCL (C1(0:M),C2(0:M)) FIXED BINARY;
                /* THE IT'H SYMBOL OCCURS C1(I) TIMES AS LEFT   */
                /*  AND C2(I) TIMES AS RIGHT                    */
        DCL B1(0:N) BIT(1), B2(0:N) BIT(1);
                /*  B(K) MEANS THAT THE K'TH PRODUCTION HAS BEEN  */
                /* ELIMINATED          */
        DCL (S0(0:N),SL(0:N),SR(0:N)) FIXED BINARY;
        DO I=1 TO M;  C1(I)=0;  C2(I)=0;   END;
BA:     DO K=1 TO N;
            S0(K)=PRD(K,0);  SL(K)=PRD(K,1);  J=5;
            DO WHILE (PRD(K,J)=0);  J=J-1;  END;
            SR(K)=PRD(K,J);  B1(K)='1'B;  B2(K)='1'B;
            C2(S0(K))=C1(S0(K))+1;   C1(S0(K))=C2(S0(K));
            END BA;
        DO I=1 TO MM;
            DO J=1 TO M;  R(I,J)='0'B;  L(I,J)='0'B;  END;   END;
        NN=N;   CHANGE='1'B;
BB:     DO WHILE (CHANGE & NN>0);
            CHANGE='0'B;
            DO K=1 TO N;
                IF B1(K) THEN DO;
                    A=S0(K);  B=SL(K);
                    IF ¬ L(A,B) THEN DO;  L(A,B)='1'B; CHANGE='1'B;
                        END;
                    IF B<=MM THEN DO J=1 TO M;
                        IF ¬ L(A,J) THEN IF L(B,J) THEN DO;
                            L(A,J)='1'B;  CHANGE='1'B; END; END;
                IF C1(B)=0 THEN DO; B1(K)='0'B; C1(A)=C1(A)-1;
                    NN=NN-1;   END;
            END BB;
        NN=N;   CHANGE='1'B;
BC:     DO WHILE (CHANGE & NN>0);
            CHANGE='0'B;
            DO K=1 TO N;
                IF B2(K) THEN DO;
                    A=S0(K);  B=SR(K);
                    IF ¬ R(A,B) THEN DO; R(A,B)='1'B; CHANGE='1'B;END;
                    IF B<=MM THEN DO J=1 TO M;
                        IF ¬ R(A,J) THEN IF R(B,J) THEN DO;
                            R(A,J)='1'B;  CHANGE='1'B;  END; END;
                    IF C2(B)=0 THEN DO;
                        B2(K)='0'B; C2(A)=C2(A)-1; NN=NN-1;  END;
            END BC;
        /* OUTPUT LEFT AND RIGHT SYMBOL SETS    */
        IF CHOICE(4) THEN   AD:  DO;
            PUT EDIT('RIGHT SYMBOL SETS')(PAGE,A);
            DO I=1 TO MM;
                PUT EDIT(SYT(I),'::=')(SKIP,2 A);
                DO J=1 TO M;
                    IF R(I,J) THEN PUT EDIT(SYT(J))(X(2),A);  END;
```

```
            END;
            PUT EDIT('LEFT SYMBOL SETS')(PAGE,A);
            DO I=1 TO MM;
                PUT EDIT(SYT(I),'::=')(SKIP,2 A);
                DO J=1 TO M;  IF L(I,J) THEN PUT EDIT(SYT(J))(X(2),A);
                        END;
            END;
            END AD;
        END STEPB;
STEPC:  BEGIN;
        DCL (U,V,P,Q) FIXED BINARY;
ENTER:  PROC (X,Y,S);
            DCL T CHAR(1);
            DCL (X,Y) FIXED BINARY,S CHAR(1);
            T=H(X,Y);
            IF T¬=' ' & T¬=S THEN DO;
                IF ¬ ERRORFLAG THEN PUT LIST('HINTS REGARDING ERRORS')
                    PAGE;
                ERRORFLAG='1'B;
                PUT EDIT(U,SYT(X),T,S,SYT(Y))(SKIP,F(4),X(2),4 A);
                END;
            H(X,Y)=S;
            IF T¬=S THEN DO;
                        HM(X,HM(X,0))=Y;  HM(X,0)=HM(X,0)+1;
                END;
            IF HM(X,0)>30 THEN PUT EDIT('HM MATRIX FULL AT ROW ',X)
                    (SKIP,A,F(4));
            END ENTER;
FAIL:   PROC (U,V);
            DCL (U,V) FIXED BINARY;
            PUT EDIT('NO PRECEDENCE FUNCTIONS EXIST',SYT(U),SYT(V))
                 (SKIP,3(A,X(4)));
            GO TO FIN;
            END FAIL;
FIXUPROW:   PROC(I,L,X)  RECURSIVE;
            DCL (I,L,X,J) FIXED BINARY;
            DCL (A,B) FIXED BINARY;
            A=1;   B=0;
            F(I)=G(L)+X;
            IF K1=K THEN DO;
                IF H(I,K)='<' & F(I)>=G(K) | H(I,K)='=' & F(I)¬=G(K)
                    THEN CALL FAIL(I,K);  END;
            DO J=K1 TO 1 BY -1;
                IF H(I,J)='<' & F(I)>=G(J) THEN CALL FIXUPCOL(I,J,A);
                    ELSE IF H(I,J)='=' & F(I)¬=G(J) THEN
                        CALL FIXUPCOL(I,J,B);  END;
            RETURN;    END FIXUPROW;
FIXUPCOL:   PROC(L,J,X) RECURSIVE;
            DCL (L,J,X,I) FIXED BINARY;
            DCL (A,B) FIXED BINARY;
            A=1;   B=0;
            G(J)=F(L)+X;
            IF K1¬=K THEN DO;
                IF H(K,J)='>' & F(K)<=G(J) |H(K,J)='=' & F(K)¬=G(J)
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,

                    THEN CALL FAIL(K,J);    END;
          DO I=K TO 1 BY -1;
               IF H(I,J)='>' & F(I)<=G(J) THEN CALL FIXUPROW(I,J,A);
                    ELSE IF H(I,J)='=' & F(I)¬=G(J) THEN
                         CALL FIXUPROW(I,J,B);    END;
          RETURN;    END FIXUPCOL;
     /*      FIND    H   PRECEDENCE MATRIX      */
     DO I=0 TO M; DO J=0 TO M; H(I,J)=' ';END; END;
     DO I=1 TO M;  HM(I,0)=1; DO J=1 TO 30; HM(I,J)=0; END; END;
UV:  DO U=1 TO N;
          DO V=2 TO 5;
               IF PRD(U,V)¬=0 THEN DO;
                    P=PRD(U,V-1); Q=PRD(U,V); CALL ENTER(P,Q,'=');
                    IF P<=MM THEN DO;
                         DO I=1 TO M;
                              IF R(P,I) THEN CALL ENTER(I,Q,'>'); END;
                    IF Q<=MM THEN DO J=1 TO M.
                         IF L(Q,J) THEN DO;
                              CALL ENTER(P,J,'<');
                              DO I=1 TO M;
                                   IF R(P,I) THEN CALL ENTER(I,J,'>');
                                   END;
                              END;
                         END;
                    END;
                    ELSE IF Q<=MM THEN DO J=1 TO M;
                         IF L(Q,J) THEN CALL ENTER(P,J,'<'); END;
          END UV;
     IF CHOICE(5) THEN    AF:  DO;
          PUT EDIT('PRECEDENCE MATRIX')(PAGE,A);
          PUT EDIT ((J/10 DO J=10 TO M BY 10))(SKIP,X(6),5(X(10),F(2)
               ));
          DO I=1 TO M;
               PUT EDIT(I,' ')(SKIP,F(4),X(1),A);
               DO J=0 TO M BY 10;
                    IF M>J+9 THEN U=J+9; ELSE U=M;
                    PUT EDIT((H(I,K) DO K= J TO U),'.')(11 A);
                    END;
               END;
          END AF;
     IF ERRORFLAG THEN DO;
          PUT EDIT('PRECEDENCE VIOLATION OCCURRED')(SKIP,A);
          GO TO  FIN;     END;
     PUT EDIT('NO PRECEDENCE VIOLATIONS OCCURRED')(SKIP,A);
     /*    PUNCH     HM        */
     IF CHOICE(8) THEN DO;
          I=0;A=15; B=4; P=0;  V=30;  COM='HM'; CALL COMPNCH(COM);
          DO U=1 TO M; CALL INTPNCH(HM(U,*),I,XM,P,V,A,B);  END;
                    END;
     /*  BUILD F AND G PRECEDENCE FUNCTIONS     */
     IF ¬ CHOICE(6) THEN GO TO  FIN;
     /*   INITIALIZE        */
     K1=0;  DO I=0 TO M; F(I)=0; G(I)=0; END;
     A=1;    B=0;
```

```
PROCEDURE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,

KKLOOP: DO K=1 TO M;
            FMIN=1;
            DO J=1 TO K1;
                IF H(K,J)='>' & FMIN<=G(J) THEN FMIN=G(J)+1;
                    ELSE IF H(K,J)='=' & FMIN<G(J) THEN FMIN=G(J); END;
            F(K)=FMIN;
            DO J=K1 TO 1 BY -1;
                IF H(K,J)='<' & FMIN>=G(J) THEN CALL FIXUPCOL(K,J,A);
                    ELSE IF H(K,J)='=' & FMIN>G(J) THEN
                            CALL FIXUPCOL(K,J,B);   END;
            K1=K1+1;      GMIN=1;
            DO I=1 TO K;
                IF H(I,K)='<' & F(I)>=GMIN THEN GMIN=F(I)+1;
                    ELSE IF H(I,K)='=' & F(I)>GMIN THEN GMIN=F(I); END;
            G(K)=GMIN;
            DO I=K TO 1 BY -1;
                IF H(I,K)='>' & F(I)<=GMIN THEN CALL FIXUPROW(I,K,A);
                    ELSE IF H(I,K)='=' & F(I)<GMIN THEN
                        CALL FIXUPROW(I,K,B);  END;
            END KKLOOP;
        PUT EDIT('PRECEDENCE FUNCTIONS F AND G')(PAGE,A);
        DO I=1 TO M;
            PUT EDIT(I,SYT(I),F(I),G(I))(SKIP,F(3),X(1),A,2 F(6));
        END;
        /*  PUNCH   F   AND   G    */
        IF CHOICE(8) THEN APPP:   DO;
            I=0;  J=XM ;   A=15;   B=4;
            COM='F';   CALL COMPNCH(COM);
            F(M+1)=0;  G(M+1)=0;  K=M+1;
            CALL INTPNCH(F,I,J,I,K,A,B);
            COM='G';   CALL COMPNCH(COM);
            CALL INTPNCH(G,I,J,I,K,A,B);
            COM='END';  CALL COMPNCH(COM);
            END APPP;
        END STEPC;
FIN:
            PUT LIST('FINISH TIME=',TIME) SKIP;
            END ANALYZE;
```

```
PROCEDURE OPTIONS(MAIN):


 PARSER: PROCEDURE OPTIONS(MAIN);
        DCL N FIXED BINARY INITIAL(50 );/*NUMBER OF PRODUCTIONS*/
         DCL M FIXED BINARY INITIAL(50 );/* NUMBER OF SYMBOLS*/
         DCL MM FIXED BINARY INITIAL(50 );/*NO. NON-BASIC SYMBOLS*/
         DCL XN FIXED BINARY;
         GET LIST(M,N,MM);
 BLOCK:    BEGIN;
         DCL (I,J,K,L,KK) FIXED BINARY,
         DCL (I1,I2) FIXED BINARY;
         DCL S(0:50) FIXED BINARY;          /*  PARSING STACK    */
             DCL VS(0:50) CHAR(400) VAR;     /*  VALUE STACK   */
         DCL QUOTE BIT(1);     /*BOOLEAN FOR QUOTING BASIC SYMBOLS */
         DCL ERSCAN CHAR(12) VAR;      /*DIAGNOSTIC TERMINATING SYMBOL */
         DCL (SYMA,SYMB) CHAR(12) VAR;  /* STACK ERROR RECOVER   */
         DCL SYM FIXED BINARY;
         DCL SYMS CHAR(80)  VARYING;
         DCL ERROR BIT(1);
         DCL HM(M,0:30) FIXED BINARY;  /* MATRIX FOR DIAGNOSTICS   */
         DCL SYT(0:M) CHAR(12);  /* SYMBOLS    */
         DCL (XWORD,XNUM,XSEQ) FIXED BINARY;
         DCL QUOTES CHAR(12);
         DCL ANS  FIXED BINARY;
         DCL A CHAR(100) VAR;
             DCL OS CHAR(80) VARYING;
             DCL F(0:M  ) FIXED BINARY;
             DCL G(0:M  ) FIXED BINARY;
             DCL KEY(0:M  ) FIXED BINARY;
             DCL PRTB(0:5*N) FIXED BINARY;
             DCL BASVAL(M  ) FIXED BINARY;
             DCL BASSYM (M  ) CHAR(12);
         DCL EQ(N  ) CHAR(12);
         DCL STACKOK INTERNAL ENTRY     RETURNS(BIT(1));
 SETUP:  PROCEDURE;
             /*  SETUP PARSING TABLES    IF FIRST CARD=ANALYZE THEN  */
         /*  INPUT IS BNF SYNTAX    IF TABLE THEN INPUT TABLES FROM  */
             /*  DATA CARDS      */
             DCL COM CHAR(100) VAR;
             PUT LIST('SETUP') PAGE;
             GET LIST(COM);
             PUT LIST(COM) SKIP;
             IF COM='TABLE' THEN DO;
                 CALL TABREAD; GET LIST(ERSCAN,SYMA,SYMB);
                 CALL TABPRINT;
                 RETURN;   END;
             ELSE IF COM='ANALYZE' THEN DO;
                 CALL ANALYZE(M,MM,N,F,G,KEY,PRTB,QUOTES,XNUM,XWORD,
                              BASVAL,BASSYM,EQ,XSEQ,ERROR,HM,SYT);
                 IF ERROR THEN PUT EDIT('PRECEDENCE ERROR')(SKIP,A);
             GET LIST(ERSCAN,SYMA,SYMB); CALL TABPRINT; RETURN; END;
             ELSE PUT EDIT('INPUT FORMAT ERROR')(SKIP,A);
             END SETUP;
 TABREAD:  PROCEDURE;
             /*   READ PARSING TABLES     */
             DCL COM CHAR(100) VAR;
```

```
PROCEDURE OPTIONS(MAIN);

            PUT LIST('TABREAD') SKIP;
LOOP:       GET LIST(COM);
            PUT LIST(COM) SKIP;
            IF COM='END' THEN RETURN;
                 IF COM='MNMM' THEN GET LIST(M,N,MM);
            IF COM='XNUM-XWORD-XSEQ' THEN GET LIST(XNUM,XWORD,XSEQ);
                 IF COM='QUOTES' THEN GET LIST(QUOTES);
                 IF COM='EQUATIONS' THEN GET LIST((EQ(J) DO J=1 TO N));
                 IF COM='BASSYM' THEN GET LIST((BASSYM(J) DO J=1 TO
                      M-1-MM));
                 IF COM='BASVAL' THEN GET LIST((BASVAL(J) DO J=1 TO
                  M-1-MM));
                 IF COM='KEY' THEN GET LIST((KEY(J) DO J=0 TO M+1));
                 IF COM='PRTB' THEN GET LIST((PRTB(J) DO J=0 TO
                KEY(M+1)));
                 IF COM='F' THEN GET LIST((F(J) DO J=0 TO M+1));
                 IF COM='G' THEN GET LIST((G(J) DO J=0 TO M+1));
            IF COM='HM' THEN DO I=1 TO M;
                GET LIST((HM(I,J) DO J=0 TO 30)); END;
            IF COM='SYMBOLS' THEN GET LIST((SYT(J) DO J=0 TO M));
            GO TO LOOP;
            END TABREAD;
TABPRINT:   PROCEDURE;
            /*  PRINT PARSING TABLES    */
            PUT EDIT('TABLE ENTRIES')(PAGE,A);
            PUT EDIT('M=',M,'N=',N,'MM=',MM)(3(X(6),A,F(6)));
        PUT EDIT('ERSCAN=',ERSCAN ,' SYMA=',SYMA,' SYMB=',SYMB)(SKIP,
                 6 A);
            PUT EDIT('XWORD=',XWORD,'XNUM=',XNUM,
                'XSEQ=',XSEQ,'QUOTES=',QUOTES)(SKIP,3(X(3),A,F(4)),
                     X(3),2 A);
            PUT EDIT('F')(SKIP,A);
            PUT EDIT((F(J) DO J=0 TO M+1))
                (100(SKIP,10(X(3),F(4))));
            PUT EDIT('G')(SKIP,A);
            PUT EDIT((G(J) DO J=0 TO M+1))
                (100(SKIP,10(X(3),F(4))));
            PUT EDIT('KEY')(SKIP,A) ;
            PUT EDIT((KEY(J) DO J=0 TO M+1))
                (100(SKIP,10(X(3),F(4))));
            PUT EDIT('PRTB')(SKIP,A);
            PUT EDIT((PRTB(J) DO J=0 TO KEY(M+1)))
                (100(SKIP,10(X(3),F(4))));
            PUT EDIT('BASVAL')(SKIP,A);
            PUT EDIT((BASVAL(J) DO J=1 TO M-1-MM))
                (100(SKIP,10(X(3),F(4))));
            PUT EDIT('BASSYM')(SKIP,A);
           PUT EDIT((BASSYM(J) DO J=1 TO M-1-MM))(100(SKIP,6(X(5),A)));
            PUT EDIT('EQUATIONS')(SKIP,A);
            PUT EDIT((EQ(J) DO J=1 TO N))          (100(SKIP,6(X(5),A)));
            PUT LIST('SYT') SKIP;
            PUT EDIT((SYT(J) DO J=0 TO M))(100(SKIP,6(X(5),A)));
            PUT LIST('HM') SKIP;
            DO I=1 TO M;
```

```
PROCEDURE OPTIONS(MAIN);

                PUT EDIT((HM(I,J) DO J=0 TO 30))(100(SKIP,20(X(2),F(4))
                                    ));
            END;
            RETURN;
            END TABPRINT;
LOOK:   PROCEDURE(OS,A,I,T);
            /*          FREE FIELD READ PROCEDURE T= TRUE IF CURRENT */
            /* INPUT STRING IS NOT A NUMBER FALSE IF PHRASE  STRING */
            /*  RETURNED IN OS    */
            /* NUMBER=DIGIT|NUMBER DIGIT|NUMBER .|(+|-) DIGIT   */
            /* LETTER=A|B|...|Z          */
            /*  WORD=LETTER|WORD LETTER   */
            /* PHRASE=WORD|SPECIAL    */
            /*  SPECIAL=ALL SYMBOLS OTHER THAN LETTERS OR DIGITS*/
            /*  IF LOWER CASE AVAILABLE CHANGE A TO LOWER CASE A */
NEXT:   PROCEDURE CHAR(1);
            /*GETS NEXT CHARACTER FROM INPUT    */
            IF I>LENGTH(A) THEN DO;
                GET EDIT(A)(A(80));
                PUT LIST(A) PAGE;
                A=A||' ';
                I=1;
            END;
            RETURN(SUBSTR(A,I,1));
            END NEXT;
CON:    PROCEDURE;
            OS=OS||SYM;    I=I+1;
            END CON;
            DCL SYM CHAR(1);
            DCL T BIT(1);
            DCL A CHAR(100) VAR;    /*READ BUFFER   */
            DCL I FIXED BINARY;  /* READ BUFFER POINTER    */
            DCL OS CHAR(80) VARYING;  /* OUTPUT STRING  */
            DCL NEXT INTERNAL ENTRY RETURNS (CHAR(1));
            DCL CON INTERNAL ENTRY;
            SYM=NEXT;  OS='';
            DO WHILE (SYM=' ');
                I=I+1;  SYM=NEXT;    END;
            IF  ((SYM>='A')&(SYM<='Z')) THEN DO;
                DO WHILE ((SYM>='A')&(SYM<='Z'));
                    CALL CON;  SYM=NEXT;  END;
                T='1'B;  RETURN;  END;
            CALL CON;
            IF (((SYM>'Z')|(SYM='+')|(SYM='-'))&
                ((NEXT='.')|(NEXT>'Z'))) THEN DO;
                DO WHILE ((NEXT>'Z')|(NEXT='.'));
                    SYM=NEXT;  CALL CON;    END;
                         T='0'B;       RETURN;
                END;
            IF SYM>'Z' THEN DO; T='0'B; RETURN ; END;
            T='1'B;  RETURN;
            END LOOK;
ASSIGN: PROCEDURE (QUOTE,OS,V) RECURSIVE;
            /*ASSIGNS A NUMERICAL VALUE TO CURRENT INPUT STRING */
```

```
PROCEDURE OPTIONS(MAIN);

            DCL QUOTE BIT (1);
            DCL OS CHAR(80) VARYING;
            DCL (V,J) FIXED BINARY;
            DCL T BIT(1);
            IF QUOTE THEN DO;
                CALL LOOK(OS,A,I,T);
              IF OS=QUOTES THEN  DO;
                    QUOTE='0'B;  CALL ASSIGN(QUOTE,OS,V); RETURN;
                    END;
                    V=XWORD;   RETURN;
                END;
            CALL LOOK(OS,A,I,T);
            IF T THEN DO;
                IF OS=QUOTES THEN DO;
                    QUOTE='1'B; CALL ASSIGN(QUOTE,OS,V);  RETURN; END;
                DO J=1 TO M-1-MM;
                    IF OS= BASSYM(J) THEN DO;
                        V=BASVAL(J); RETURN;
                        END;
                    END;
                    V=XWORD;   RETURN;
                END;
            V=XNUM;  RETURN;
            END ASSIGN;
ERRORREC:   PROCEDURE;
            DCL X CHAR(80) VAR;  DCL (V,W) FIXED BINARY;
            DCL T(100) FIXED BINARY;
            DCL ER BIT(1);
            I=1;              ER='1'B;      QUOTE='0'B; X=' '; K=0;
        PUT EDIT('SYNTAX ERROR--- IF MORE INPUT REQUESTED BEFORE END ',
                    'OF SYNTAX ANALYSIS THEN PLEASE ENTER (',ERSCAN,
                    ')')(SKIP, 4 A);
            DO V=1 TO M; IF SYMA=SYT(V) THEN W=V; END;
        IF S(1)¬= W THEN S(1)=W;
        DO V=1 TO M; IF SYMB=SYT(V) THEN W=V; END;
            IF S(2)¬=W THEN DO; J=0; SYM=S(1); END;
                ELSE DO;  J=2;  SYMS=ERSCAN;
                    DO V=1 TO M; IF ERSCAN=SYT(V) THEN SYM=V; END; END;
            DO WHILE (X¬=ERSCAN); K=K+1; CALL ASSIGN(QUOTE,X,T(K));END;
            QUOTE='0'B;
            A=SUBSTR(A,I+1);
            DO V=1 TO K-1;
                W=1;
                DO WHILE (HM(T(V),W)¬=T(V+1) & W<HM(T(V),0));
                    W=W+1;   END;
                IF W>=HM(T(V),0) THEN    DO;
                        PUT EDIT(SYT(T(V)),' MAY NOT BE FOLLOWED BY ',
                                SYT(T(V+1)))(SKIP,3 A); ER='0'B; END;
                END;
            IF ER THEN PUT LIST('ERROR NOT IN CURRENT INPUT') SKIP;
            PUT LIST('END OF SYNTAX ERROR ANALYSIS') SKIP;
            RETURN;
            END ERRORREC;
STACKOK:    PROCEDURE       BIT(1);
```

```
PROCEDURE OPTIONS(MAIN).

        DCL (TRUE,FALSE) BIT(1);
        DCL  W FIXED BINARY;
            TRUE='1'B;  FALSE='0'B;  W=1;
            IF S(J-1)=0 THEN RETURN(TRUE);
            DO WHILE (HM(S(J-1),W)¬=S(J)); W=W+1;  END;
            IF W>=HM(S(J-1),0) THEN RETURN(FALSE); ELSE RETURN(TRUE);
            END STACKOK;
        XN=N;
        CALL SETUP;
        PUT LIST('START OF PARSING') PAGE;
        IF ERROR THEN GO TO FINIS;
            /*  PARSING SECTION    */
        DO J=0 TO 50;       S(J)=0;  END;
PROCES:
        ANS=0;  ERROR='0'B;
        J=0;  I=81;                        QUOTE='0'B;
        CALL ASSIGN(QUOTE,SYMS,SYM);
        DO WHILE (SYM>0);
            J=J+1;  K=J;  S(J)=SYM; VS(J)=SYMS;
            CALL ASSIGN(QUOTE,SYMS,SYM);
         DO WHILE (F(S(J))>G(SYM));
            IF S(J)=XSEQ THEN GO TO PROCES;
            DO WHILE ((F(S(J-1))=G(S(J)))&(J>1)).
                J=J-1;  END;
            L=KEY(S(J));
            IF STACKOK THEN DO;
            DO WHILE (PRTB(L)¬=0.);
                KK=J+1;
                DO WHILE ((KK<=K)&(S(KK)=PRTB(L)));
                    KK=KK+1;  L=L+1;  END;
                IF ((KK>K)&(PRTB(L)<0)) THEN DO;
                    I1=J;  I2=K;
                    CALL SEMANT   (-PRTB(L),VS,I1,I2,EQ,ANS,ERROR,XN);
                    IF I1¬=J|I2¬=K THEN PUT LIST('POINTER ERROR') SKIP;
                    S(J)=PRTB(L+1);  L=0;  END;
                ELSE DO;
                    DO WHILE (PRTB(L)>0);
                        L=L+1;  END;
                    L=L+2;    END;
                END;
                IF L¬=0 THEN DO;
                      L=0;   J=0;
                        CALL ERRORREC;     END;
                END;
        ELSE DO;  J=0; CALL ERRORREC;                END;
                K=J;
            END;
         END;
        END BLOCK;
FINIS:
            END PARSER;
```