A TRANSLATOR WRITING SYSTEM
FOR SIMPLE PRECEDENCE GRAMMARS

A THESIS

PRESENTED TO

THE FACULTY OF GRADUATE STUDIES AND RESEARCH

THE UNIVERSITY OF MANITOBA

IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

IN THE DEPARTMENT OF COMPUTER SCIENCE

---

by

Bruce Foulkes

March 1971

ABSTRACT

A System (consisting of three programs written in PL/1) is described which serves as an aid in the development of Simple Precedence grammars.

A program is provided which tests a given grammar to see if it conforms to the rules of Simple Precedence.

Implementation consists of providing the semantics in the form of a PL/1 external procedure to run in conjunction with a program of the System which performs the syntactic analysis.

Any Simple Precedence grammar which meets a few restrictions imposed by the System can be implemented using this scheme.

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to Professor J. Wells for his guidance and supervision during the preparation of this thesis.

I would also like to extend my sincere thanks to Dr. P. King for his comments and criticism on this thesis.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The thesis is divided into four chapters. The current chapter outlines the objectives of the thesis, and presents a description of Simple Precedence grammars. Chapter 2 can be considered as a 'User's Manual' and describes the details of the System from the user's point of view. Chapter 3 gives a short explanation of some of the novel or difficult points in the actual programs of the System. Finally Chapter 4 presents some conclusions on this approach.

1.1 OBJECTIVES

The motivation behind this approach was the desire to provide a means by which students could design and implement their own computer languages, and thereby gain valuable experience with a minimum of time and effort.

An attempt was made to maximize both efficiency and versatility so that it would not be unreasonable to develop compilers using this system for use in limited applications.

1.2 THEORY

A phrase structure grammar can be defined by $G = (V,S,T,Z)$ where V is the total vocabulary, whose elements are called symbols and will be represented by capital letters; S is a set of syntactic rules of the form $A \rightarrow b$ where $b \neq A$, $A \in V-T$, $b \in V^*$ (where V^* is the set of all strings over V) and A is called the left part and b the right part of this syntactic rule or production; T is the set of terminal symbols; Z is a unique symbol appearing only on the left side of the set of syntactic rules.

The left-most symbol of b is referred to as a left derivable symbol of A, and is a member of the set L(A). Likewise R(A) represents all the right derivable symbols of A. If the left/right derivable symbol of A in any production is a nonterminal then all the left/right derivable symbols of it are also considered left/right derivable symbols of A.

Wirth and Weber (1) define three possible precedence relations which can occur between symbol pairs. Using $A \rightarrow xXYy$ as a sample production, the relationships can be demonstrated as follows:

1. $X \doteq Y$

2. $X \lessdot B$ if $B \in L(Y)$

3. $M \gtrdot N$ if $M \in R(X)$ and (N is Y or $N \in L(Y)$).

If there is at most only one of these precedence relationships between any two symbols of the syntax, then the grammar is referred to as a Simple Precedence grammar.

These relationships can be stored in a Precedence Matrix of size N X N where N is the total number of symbols (both terminal and nonterminal) which appear in the vocabulary. The relationships between two symbols can then be found by locating the relationship corresponding to the row of the first symbol, and the column of the second. A blank indicates that no precedence relationship exists between the two symbols in question and therefore the second symbol can never follow the first according to the grammar.

A method of storing the precedence relations in 2 X N locations instead of N X N locations as with the precedence matrix was first suggested by Robert W. Floyd (2) for Operator Precedence grammars. This was extended to Simple Precedence grammars by Wirth and Weber. The method consists in defining integer functions F and G as follows:

If $A \doteq B$ then $F(A) = G(B)$

If $A \lessdot B$ then $F(A)$ is less than $G(B)$

If $A \gtrdot B$ then $F(A)$ is greater than $G(B)$.

One disadvantage to the F and G functions is that they do not always exist. In addition, if they do exist they will always show a relationship between any two symbols while in fact the Precedence Matrix might show that none existed for some pairs of symbols.

1.2.1 PARSING TECHNIQUE

The canonical parse proceeds from left to right in a sentence written in a Simple Precedence language, reducing left-most reducible substrings as it encounters them. The symbol before the leftmost reducible substring yields precedence to its left-most symbol (i.e., has relation "$\lessdot$"), while its right-most symbol has precedence over the symbol following (i.e., has relation "$\gtrdot$"). All of the symbols in the reducible substring are of equal precedence. When the parse encounters the situation

$$A \lessdot S_o \doteq S_1 \doteq . . \doteq S_n \gtrdot B$$

then the string S_o to S_n is the right part of some production and is

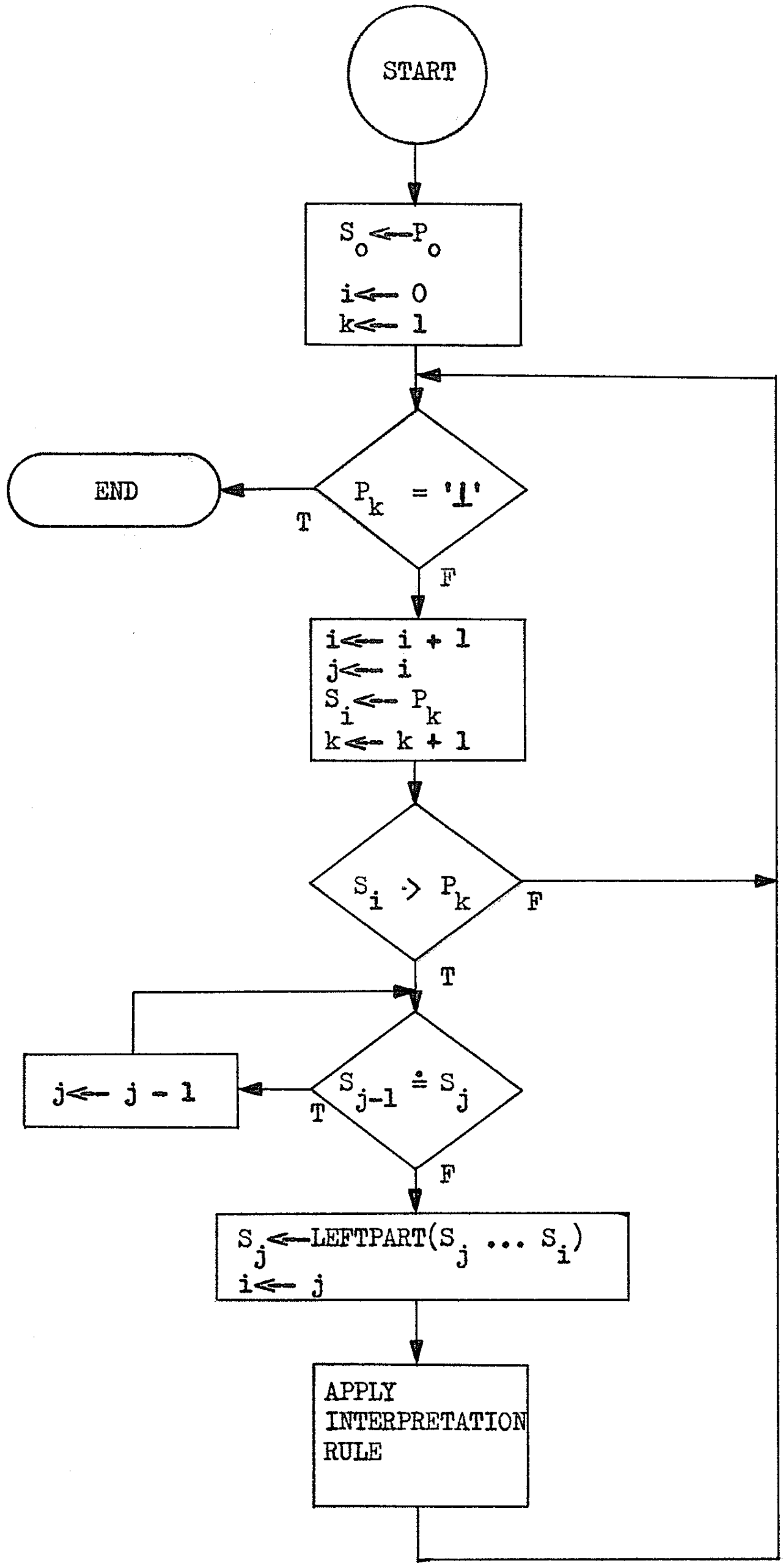
START
$S_o \Leftarrow P_o$
$i \Leftarrow 0$
$k \Leftarrow 1$
END
$P_k = '\perp'$
T
F
$i \Leftarrow i + 1$
$j \Leftarrow i$
$S_i \Leftarrow P_k$
$k \Leftarrow k + 1$
$S_i > P_k$
F
T
$j \Leftarrow j - 1$
$S_{j-1} \doteq S_j$
T
F
$S_j \Leftarrow$ LEFTPART$(S_j \ldots S_i)$
$i \Leftarrow j$
APPLY
INTERPRETATION
RULE

Fig. 1

replaced by the corresponding left part and the parse continues until only the symbol Z, the unique symbol appearing only on the left side of a production, remains signalling a successful completion of the parse.

If no two right parts of productions in the grammar are identical then the parse is unique.

The parse fails if it encounters two adjacent symbols with no precedence relation defined between them, or if it finds a left-most reducible substring which is not the right part of some production. Unless some mechanism for error recovery is built into the parse, the parse must terminate at this point with an indication that the string is not a sentence in the grammar.

Fig. 1 is a flow chart of a parsing algorithm proposed by Wirth and Weber (1). The input string consists of $P_1 \ldots P_n$. k is the index of the last symbol to be scanned. The left-most reducible substring is $S_j \ldots S_i$. The algorithm assumes the symbol $\perp$ appears before and after the input string, where for any symbol $S \in V$ $\perp \lessdot S$ and $S \gtrdot \perp$. The function LEFTPART locates the left-most reducible substring among the right parts of the productions of the language and returns the corresponding left part.

1.2.2 SEMANTICS

To add semantics to a parser for a Simple Precedence grammar one can associate an interpretation rule with every syntax rule or production in the grammar. Application of these interpretation rules in the order in which the reductions (replacing the right part of productions by the corresponding left part) were made while parsing the input results in a unique meaning of the input string being determined.

Fig. 2a shows an example of a Simple Precedence grammar which defines a binary number (i.e., a string of 1's and 0's). The interpretation rules shown with the productions will develop the decimal equivalent of any binary number defined using this grammar if applied in the correct order. For convenience the symbols BOOL, NUMBER, and DIGIT are replaced by B, N and D respectively in Figures 2b and 3. Fig. 2b is the Precedence Matrix for this Simple Precedence grammar. Fig 3 demonstrates a parse of the input string 1001.

PRODUCTIONS			INTERPRETATION RULES
1.	BOOL	::= NUMBER	null
2.	NUMBER	::= DIGIT	Total ← Value
3.		::= NUMBER DIGIT	Total ← Total X 2 + Value
4.	DIGIT	::= 0	Value ← 0
5.		::= 1	Value ← 1

Fig. 2a

	B	N	D	0	1
B					
N			≐	<·	<·
D			·>	·>	·>
0			·>	·>	·>
1			·>	·>	·>

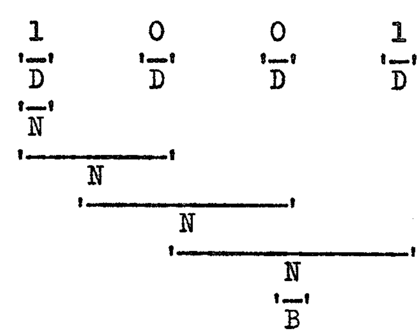

Fig. 3

From this it can be seen that the interpretation rules should be applied in the order 5,2,4,3,4,3,5,3,1. This will result in the correct answer 9, being produced in Total.

CHAPTER 2

USER'S GUIDE

The System consists of three programs written in PL/1. The first of these programs, TESTPREC, tests Simple Precedence grammars. The other two programs, EXTRACT and ANALYSE, are used in the implementation of newly defined languages. EXTRACT reads the Simple Precedence grammar and creates a file called TABLES containing information about the language to be implemented. ANALYSE then uses the information in TABLES to perform a lexical scan and syntactic analysis on a source program written in the new language. It remains the responsibility of the person implementing the language to provide the semantics. This is accomplished by writing a PL/1 external procedure INTERPRET which is called by ANALYSE. Fig. 4 demonstrates the approach used.

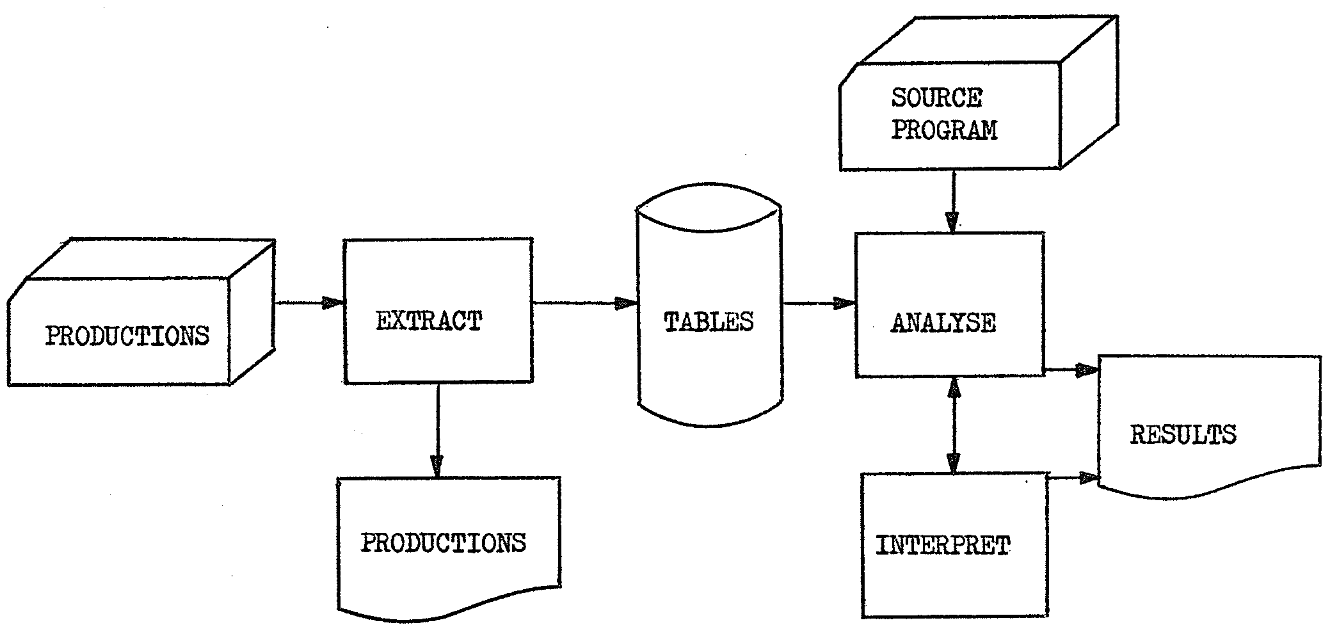

Fig. 4

The three programs of the System are presented in the order in which they are used, and a sample language is shown at various stages of development.

2.1 TESTPREC

Purpose of the Program. The program is designed to determine if a given grammar is a Simple Precedence Phrase Structure grammar.

Input. One production is given on each data card. The symbol on the left side of the production must start in column 1. The blank is the delimiter between symbols. (i.e., between the left symbol and the right-hand side of the production, and between each of the symbols on the right-hand side of the production.) Any number of blanks may be inserted between symbols. If the left-hand side of a production is the same as that of the previous production, it may be omitted by leaving column 1 blank, and the previous left part will be assumed. The right part may then start anywhere from column 2 on. Fig. 5 shows the sample language input to both TESTPREC and EXTRACT.

Output.

1. A list of the Productions in readable form. (See Fig. 6)
2. A list of the NONTERMINAL and TERMINAL symbols used in the syntax. (See Fig. 7)
3. A list of the precedence violations (if any) which occurred and explanations of these violations.

Optional Output.

1. A printout of the PRECEDENCE MATRIX. The matrix is printed

even if precedence violations occur, but only 1 relationship is shown. (i.e., violations are not shown in the precedence matrix) Specify 'MATRIX' in the PARM list on the EXEC card. (See Fig. 8)

2. A listing of the F and G Functions if they exist. They will not be produced if a precedence violation is detected. Specify 'FUNCTIONS' in the PARM list on the EXEC card. (See Fig. 9)

Limitations.

1. There must not be more than 180 symbols in the grammar.
2. There must not be more than 6 symbols on the right-hand side of any production.

If either of these limits is exceeded an error message is printed and the program terminates abnormally.

3. A symbol is restricted to 12 characters. Longer symbols are truncated to 12 and a warning given, but the program continues normally.

Explanation of the Precedence Matrix. The symbols are in the same relative positions on both the horizontal and vertical axis. To find the relationship between two symbols, locate the first symbol in the listing at the left margin. Locate the second symbol in this listing and note the number associated with it. Proceed along the row corresponding to the first symbol to the column corresponding to the second number.

TESTPREC Error Messages. When a relationship is found between symbols of the syntax, the relationship is inserted into the precedence

```
PROGRAM START BLOCK FINISH
BLOCK BEGIN BODY END
BODY BODY-
BODY- STATLIST
STATLIST STATLIST ; STATMNT
 STATMNT
STATMENT VAR := CHOICE
 GO TO VAR
 I/O
 BLOCK
STATMNT STATMENT
 DECL
 LABDEF STATMNT
 IF-ELSE STATMNT
 IFCLAUSE STATMENT
IFCLAUSE IF RELATION THEN
IF-ELSE IFCLAUSE STATMENT ELSE
EXPR EXPR-
EXPR- EXPR- + TERM
 EXPR- - TERM
 + TERM
 - TERM
 TERM
CHOICE- EXPR
CHOICE CHOICE-
RELATION CHOICE
 CHOICE ¬= CHOICE
 CHOICE <= CHOICE
 CHOICE >= CHOICE
 CHOICE = CHOICE
 CHOICE < CHOICE
 CHOICE > CHOICE
TERM TERM-
TERM- TERM- * FACTOR
 TERM- / FACTOR
 FACTOR
FACTOR ( EXPR )
 ANYSTRING
 NUMBER
 VAR
VAR VAR_TABLE
NUMBER NUMERO
I/O WRITE VAR
 READ VAR
 I/O , VAR
DECL TYPE VAR
 DECL , VAR
TYPE INTEGER
 LABEL
LABDEF VAR :
```

Fig. 5

PRODUCTIONS

1	PROGRAM	::=	START	BLOCK	FINISH
2	BLOCK	::=	BEGIN	BODY	END
3	BODY	::=	BODY-		
4	BODY-	::=	STATLIST		
5	STATLIST	::=	STATLIST	;	STATMNT
6		::=	STATMNT		
7	STATMENT	::=	VAR	:=	CHOICE
8		::=	GO	TO	VAR
9		::=	I/O		
10		::=	BLOCK		
11	STATMNT	::=	STATMENT		
12		::=	DECL		
13		::=	LABDEF	STATMNT	
14		::=	IF-ELSE	STATMNT	
15		::=	IFCLAUSE	STATMENT	
16	IFCLAUSE	::=	IF	RELATION	THEN
17	IF-ELSE	::=	IFCLAUSE	STATMENT	ELSE
18	EXPR	::=	EXPR-		
19	EXPR-	::=	EXPR-	+	TERM
20		::=	EXPR-	-	TERM
21		::=	+	TERM	
22		::=	-	TERM	
23		::=	TERM		
24	CHOICE-	::=	EXPR		
25	CHOICE	::=	CHOICE-		
26	RELATION	::=	CHOICE		
27		::=	CHOICE	>=	CHOICE
28		::=	CHOICE	<=	CHOICE
29		::=	CHOICE	¬=	CHOICE
30		::=	CHOICE	=	CHOICE
31		::=	CHOICE	<	CHOICE
32		::=	CHOICE	>	CHOICE
33	TERM	::=	TERM-		
34	TERM-	::=	TERM-	*	FACTOR
35		::=	TERM-	/	FACTOR
36		::=	FACTOR		
37	FACTOR	::=	(	EXPR	)
38		::=	ANYSTRING		
39		::=	NUMBER		
40		::=	VAR		
41	VAR	::=	VAR_TABLE		
42	NUMBER	::=	NUMERO		
43	I/O	::=	WRITE	VAR	
44		::=	READ	VAR	
45		::=	I/O	,	VAR
46	DECL	::=	TYPE	VAR	
47		::=	DECL	,	VAR
48	TYPE	::=	INTEGER		
49		::=	LABEL		
50	LABDEF	::=	VAR	:	

Fig. 6

NONTERMINAL SYMBOLS

PROGRAM	BLOCK	BODY	BODY-	STATLIST	STATMNT
STATMENT	VAR	CHOICE	I/O	DECL	LABDEF
IF-ELSE	IFCLAUSE	RELATION	EXPR	EXPR-	TERM
CHOICE-	TERM-	FACTOR	NUMBER	TYPE	

TERMINAL SYMBOLS

START	FINISH	BEGIN	END	;	:=
GO	TO	IF	THEN	ELSE	+
-	>=	<=	¬=	=	<
>	*	/	(	)	ANYSTRING
VAR_TABLE	NUMERO	WRITE	READ	,	INTEGER
LABEL	:				

VIOLATIONS

NO PRECECENCE VIOLATIONS OCCURRED

Fig. 7

```
                                 PRECEDENCE MATRIX

                ..........|.........|.........|.........|.........|.....
 1 PROGRAM      |
 2 START        |  = <
 3 BLOCK        |   =  >  >               >
 4 FINISH       |
 5 BEGIN        |  < <= << <<<  < <<<<<<                       < << <<<
 6 BODY         |      =
 7 END          |   >  >  >               >
 8 BODY-        |      >
 9 STATLIST     |      >  =
10 ;            |  < <     =<<  < <<<<<<                       < << <<<
11 STATMNT      |      >  >
12 STATMENT     |      >  >               =
13 VAR          |      >  >   =          >>  > > >>>>>> > > >      >   =
14 :=           |            < =           <<<<<<      < < < <<<<
15 CHOICE       |      >  >              >>      ======
16 GO           |                =
17 TO           |            =                                 <
18 I/O          |      >  >               >                        =
19 DECL         |      >  >                                        =
20 LABDEF       |  < <     =<<  < <<<<<<                       < << <<<
21 IF-ELSE      |  < <     =<<  < <<<<<<                       < << <<<
22 IFCLAUSE     |  < <      =<  < <                            < <<
23 IF           |            < <        =  <<<<<<      < < < <<<<
24 RELATION     |                        =
25 THEN         |  > >      >>  > >                            > >>
26 ELSE         |  > >     >>>  > >>>>>>                       > >> >>>
27 EXPR         |      >  >              >>      >>>>>>     =
28 EXPR-        |      >  >              >>  = = >>>>>>     >
29 +            |            <                =        < < < <<<<
30 TERM         |      >  >              >>  > > >>>>>>     >
31 -            |            <                =        < < < <<<<
32 CHOICE-      |      >  >              >>      >>>>>>
33 >=           |            < =           <<<<<<      < < < <<<<
34 <=           |            < =           <<<<<<      < < < <<<<
35 ¬=           |            < =           <<<<<<      < < < <<<<
36 =            |            < =           <<<<<<      < < < <<<<
37 <            |            < =           <<<<<<      < < < <<<<
38 >            |            < =           <<<<<<      < < < <<<<
39 TERM-        |      >  >              >>  > > >>>>>> = = >
40 *            |            <                           = < <<<<
41 FACTOR       |      >  >              >>  > > >>>>>> > > >
42 /            |            <                           = < <<<<
43 (            |            <             =<<<<       < < < <<<<
44 )            |      >  >              >>  > > >>>>>> > > >
45 ANYSTRING    |      >  >              >>  > > >>>>>> > > >
46 NUMBER       |      >  >              >>  > > >>>>>> > > >
47 VAR_TABLE    |      >  >   >          >>  > > >>>>>> > > >      >   >
48 NUMERO       |      >  >              >>  > > >>>>>> > > >
49 WRITE        |            =                                 <
50 READ         |            =                                 <
51 ,            |            =                                 <
52 TYPE         |            =                                 <
53 INTEGER      |            >                                 >
54 LABEL        |            >                                 >
55 :            |  > >     >>>  > >>>>>>                       > >> >>>
                ..........|.........|.........|.........|.........|.....
```

Fig. 8

PRECEDENCE FUNCTIONS		F	G
1	PROGRAM	1	1
2	START	4	1
3	BLOCK	4	4
4	FINISH	1	4
5	BEGIN	1	5
6	BODY	1	1
7	END	5	1
8	BODY-	2	2
9	STATLIST	2	2
10	;	2	2
11	STATMNT	3	2
12	STATMENT	3	3
13	VAR	8	6
14	:=	2	8
15	CHOICE	4	2
16	GO	1	4
17	TO	6	1
18	I/O	4	4
19	DECL	4	3
20	LABDEF	2	3
21	IF-ELSE	2	3
22	IFCLAUSE	3	3
23	IF	1	3
24	RELATION	1	1
25	THEN	8	1
26	ELSE	8	3
27	EXPR	5	3
28	EXPR-	6	4
29	+	4	6
30	TERM	7	4
31	-	4	6
32	CHOICE-	5	3
33	>=	2	4
34	<=	2	4
35	¬=	2	4
36	=	2	4
37	<	2	4
38	>	2	4
39	TERM-	7	5
40	*	5	7
41	FACTOR	8	5
42	/	5	7
43	(	3	6
44	)	8	5
45	ANYSTRING	8	6
46	NUMBER	8	6
47	VAR_TABLE	9	7
48	NUMERO	8	6
49	WRITE	6	4
50	READ	6	4
51	,	6	4
52	TYPE	6	3
53	INTEGER	8	3
54	LABEL	8	3
55	:	8	8

Fig. 9

matrix. If a different relationship has already been stored in the matrix for this pair of symbols, then an error message is printed giving the relationship already in the matrix, and the second one which was to be inserted.

According to the definitions of Simple Precedence grammars, there are four possible situations which give rise to precedence relations: one each for ≐ and <· and two for ·>. If WWW, XXX, YYY and ZZZ are symbols used in the syntax of the language, then the following examples of error messages demonstrate the four possible cases.

```
XXX ≐> YYY  NOTE: = BECAUSE XXX ADJACENT TO YYY IN ###
            NOTE: > BECAUSE XXX IS RDS OF WWW  &  WWW = YYY IN @@@
```

This says that XXX has equal precedence with, and precedence over YYY. They are equal in precedence because XXX occurs adjacent to YYY in production number ###. XXX has precedence over YYY because XXX is a Right Derivable Symbol of WWW and WWW occurs adjacent to YYY in production number @@@.

```
XXX <> YYY  NOTE: < BECAUSE YYY IS LDS OF WWW  &  XXX = WWW IN ???
            NOTE: > BECAUSE XXX IS RDS OF ZZZ  &  YYY IS LDS OF WWW
            & ZZZ = WWW IN +++
```

This says that XXX yields precedence to, and has precedence over YYY. XXX yields precedence to YYY because YYY is a Left Derivable Symbol of WWW and XXX occurs adjacent to WWW in production number ???. XXX has precedence over YYY because XXX is a Right Derivable Symbol of ZZZ and YYY is a Left Derivable Symbol of WWW and ZZZ occurs adjacent to WWW in production number +++.

If the use of the equal sign in the explanations of the precedence violations is confusing, substitute "occurs to the left of, and adjacent to".

2.2 EXTRACT

Purpose of the Program. The program is designed to create a file called TABLES and store in it all of the pertinent information about the grammar. TABLES is declared in EXTRACT with the attributes FILE STREAM OUTPUT. The JCL supplied by the user for this file must be consistent with these attributes. Once this file has been created, it can be read in by ANALYSE at run time, and a syntactic analysis can then be performed upon an input stream according to the rules of this grammar.

Input. Exactly the same input is used here as for TESTPREC. The grammar should be tested by TESTPREC first to check for precedence violations.

Output.

1. The file TABLES is created on the device specified in the JCL.
2. A list of the production of the grammar and a statement saying whether TABLES was loaded successfully or not are printed on the SYSPRINT data set. See Fig. 10 for the output using the sample language.

Error Messages. The error messages, except those for precedence violations, are the same as given by TESTPREC. If a precedence violation is found an error message is given to this effect, but the violation is not listed and the program immediately terminates.

Limitations. The limitations on the input grammar are the same as for TESTPREC. If a grammar runs successfully using TESTPREC then TABLES should be loaded successfully using EXTRACT.

PRODUCTIONS

1	PROGRAM	::=	START	BLOCK	FINISH
2	BLOCK	::=	BEGIN	BODY	END
3	BODY	::=	BODY-		
4	BODY-	::=	STATLIST		
5	STATLIST	::=	STATLIST	;	STATMNT
6		::=	STATMNT		
7	STATMENT	::=	VAR	:=	CHOICE
8		::=	GO	TO	VAR
9		::=	I/O		
10		::=	BLOCK		
11	STATMNT	::=	STATMENT		
12		::=	DECL		
13		::=	LABDEF	STATMNT	
14		::=	IF-ELSE	STATMNT	
15		::=	IFCLAUSE	STATMENT	
16	IFCLAUSE	::=	IF	RELATION	THEN
17	IF-ELSE	::=	IFCLAUSE	STATMENT	ELSE
18	EXPR	::=	EXPR-		
19	EXPR-	::=	EXPR-	+	TERM
20		::=	EXPR-	-	TERM
21		::=	+	TERM	
22		::=	-	TERM	
23		::=	TERM		
24	CHOICE-	::=	EXPR		
25	CHOICE	::=	CHOICE-		
26	RELATION	::=	CHOICE		
27		::=	CHOICE	>=	CHOICE
28		::=	CHOICE	<=	CHOICE
29		::=	CHOICE	¬=	CHOICE
30		::=	CHOICE	=	CHOICE
31		::=	CHOICE	<	CHOICE
32		::=	CHOICE	>	CHOICE
33	TERM	::=	TERM-		
34	TERM-	::=	TERM-	*	FACTOR
35		::=	TERM-	/	FACTOR
36		::=	FACTOR		
37	FACTOR	::=	(	EXPR	)
38		::=	ANYSTRING		
39		::=	NUMBER		
40		::=	VAR		
41	VAR	::=	VAR_TABLE		
42	NUMBER	::=	NUMERO		
43	I/O	::=	WRITE	VAR	
44		::=	READ	VAR	
45		::=	I/O	,	VAR
46	DECL	::=	TYPE	VAR	
47		::=	DECL	,	VAR
48	TYPE	::=	INTEGER		
49		::=	LABEL		
50	LABDEF	::=	VAR	:	

TABLES LOADED SUCCESSFULLY

Fig. 10

2.3 ANALYSE

The program ANALYSE is a framework for a one-pass compiler for Simple Precedence grammars. It is general because the details about the language are read in at run time. Once TABLES has been read in ANALYSE behaves as if it has been specially written for this language described by the tables.

The reading of all the pertinent information at run time is of course, an overhead which would not be tolerated in a production compiler. At the same time a production compiler is designed to handle only one language. Tolerating this limited increase in cost on every run results in a powerful tool for experimentation in language design plus a not unreasonable tool for the implementation of languages on a somewhat smaller scale than production compilers.

ANALYSE represents only part of the implementation of a compiler. It performs the canonical parse but it is the user written PL/1 external procedure INTERPRET which provides semantics.

Whenever ANALYSE finds a left-most reducible substring, it calls INTERPRET to apply some meaning to what it has found. The necessary information is made available to INTERPRET through the use of a parameter list. When control is returned to ANALYSE a reduction is made and the parse continues.

ANALYSE can logically be divided into two parts; the Scan and the Parse.

<u>The Scan</u>. The purpose of the Scan is to read in the Source program written in the user's language and perform a lexical analysis

of it and provide this information to the Parse. Any source program consists entirely of terminal symbols as defined by the grammar. The Scan must search the input and pick out these terminal symbols.

In order to keep the scan general, but at the same time to keep its operation as efficient as possible, some basic rules were laid down (and hence some restrictions were imposed) as to what the scan would be required to recognise.

When the scan succeeds in isolating and recognising a symbol in the input stream, the symbol is replaced by a number which thereafter represents it. These symbol numbers are the ones assigned by TESTPREC. Examination of the print-out of the precedence matrix gives the number used to represent each of the symbols.

The first distinction ANALYSE makes upon encountering a non-blank character is between operators (i.e., special symbols - any character which is not a letter of the alphabet or a digit) and all others. If the character is an operator, the scan checks to see if the next position in the input also contains an operator. If it does, the Scan checks its list of all the double operators (i.e., such things as := and >=) which were used in the syntax. If this search is not successful, the first symbol is checked against the list of all single operators. A failure on this search indicates that an undefined symbol has been found.

If the character found is not an operator, a second distinction is made between letters of the alphabet and digits. If it is a letter of the alphabet it is assumed to be the start of a key-word (i.e., a terminal symbol used in the syntax starting with a letter of the alphabet - such as BEGIN or IF) or an identifier. Both key-words and identifiers are assumed by ANALYSE to start with a letter, followed by a sequence of letters or digits. The scan therefore searches until it finds a blank or operator and thus isolates the key-word or identifier. If the isolated string is less than or equal to 12 characters in length, the list of all the key-words used in the syntax is searched. If it is found then the corresponding number is sent to the Parse. If the string is longer than 12 characters, or if it was not found to be a key-word, then it is assumed to be an identifier.

If the character found is a digit then it is assumed that a number has been encountered. The user has the option of either defining the numbers used in the syntax explicitly for example:

```
NUMBER  ::= DIGIT
        ::= NUMBER   DIGIT
DIGIT   ::= 0
        ::= 1
       etc.
```

or by using a "reserved word" (see Numbers in 2.3.1) and letting ANALYSE pick out an integer number of any length on one input card. (i.e., a string of digits not including any special characters). If an explicit definition was made then the list of key-words is searched to find the internal number corresponding to that digit. The flow chart in Fig. 11 shows the basic operation of the Scan.

<u>The Parse</u>. The Parse operates in the manner described in (1), with a push-down Stack. When a left-most reducible substring is isolated at the top of the STACK the external procedure INTERPRET is called and the production number and pointers to the left-most and right-most symbols

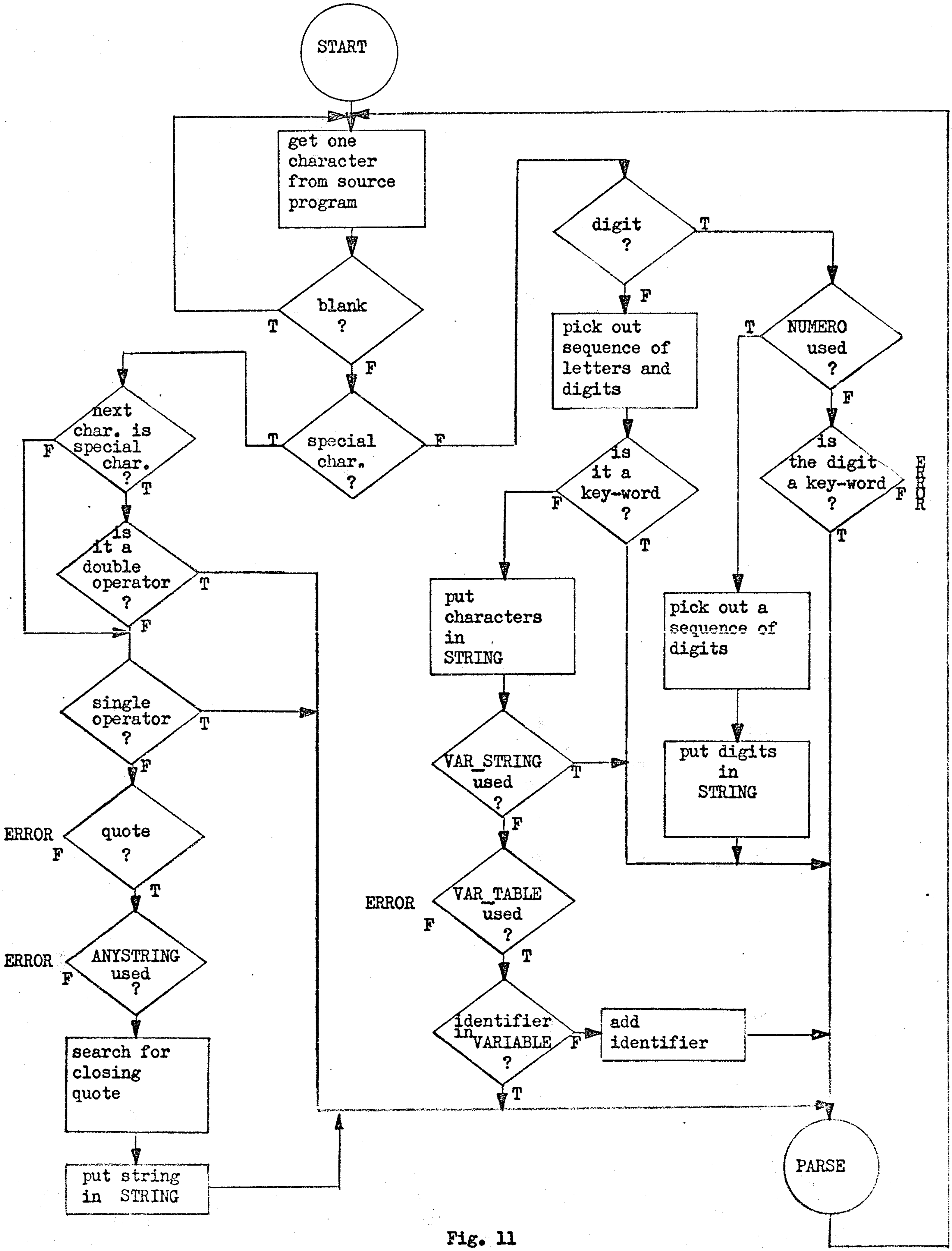
START
get one character from source program
blank ?
T
F
special char. ?
T
F
next char. is special char. ?
F
T
is it a double operator ?
T
F
single operator ?
T
F
quote ?
ERROR
F
T
ANYSTRING used ?
ERROR
F
search for closing quote
put string in STRING
digit ?
T
F
pick out sequence of letters and digits
is it a key-word ?
F
T
put characters in STRING
VAR_STRING used ?
T
F
VAR_TABLE used ?
ERROR
F
T
identifier in VARIABLE ?
F
T
add identifier
NUMERO used ?
T
F
is the digit a key-word ?
F
ERROR
T
pick out a sequence of digits
put digits in STRING
PARSE

Fig. 11

are passed. When control is returned to ANALYSE this substring is replaced by the corresponding left-part of the production and the Parse continues by comparing this with the next incoming symbol. This process continues until the Scan runs out of symbols to provide the Parse or the procedure INTERPRET terminates execution.

2.3.1 SCANNER CONVENTIONS

Identifiers. The user is not free to define his own syntax for identifiers. They must start with a letter which may be followed by any number of letters or digits, with the restriction that the name may not be split over the end of a card. To use this definition the user must use one of two possible reserved words in his syntax. This reserved word is then assigned an internal symbol number, and it is this number which the Scan substitutes for the variable name. The procedure INTERPRET must of course know which variable occurred and not just that a variable was found. This information is provided to INTERPRET in one of two ways and hence the two reserved words. These words are "VAR_STRING" and "VAR_TABLE". Descriptions of their use follows.

VAR_STRING

The Scan isolates a variable as defined above. It passes the internal symbol number assigned to it to the Parse. It then inserts the character string corresponding to the identifier into the INTERPRET parameter STRING. STRING is a character string with the varying attribute. INTERPRET can then use STRING and take the appropriate action.

VAR_TABLE

As with VAR_STRING the Scan isolates the identifier and sends the internal symbol number to the Parse. ANALYSE sets up an array called VARIABLE which can accomodate a maximum of 250 identifiers. When an identifier is found the array VARIABLE is searched to see if it had been entered previously. It it has then a number which corresponds to that subscript of VARIABLE is passed to INTERPRET. If it is a new identifier then a new entry is made and the new number is passed. This number is assigned to the INTERPRET parameter INFO. The array VARIABLE is also a parameter of INTERPRET and INTERPRET can use the number in INFO as the subscript in VARIABLE to get the character string for the identifier. The array VARIABLE only stores 12 characters of the name. Longer names are truncated to 12 significant characters. In the event that the user wants to use VAR_TABLE to build up a table of the identifiers instead of doing it himself through VAR_STRING but still wants to be able to access a complete name which is longer than 12 characters, the character string corresponding to the name is inserted in the parameter STRING the same as with VAR_STRING. Although the number INFO is passed to INTERPRET, it is still up to INTERPRET to take care of such things as local and global variables of the same name, etc.

Numbers. As previously stated, the user has the option of defining numbers in his syntax, or letting ANALYSE pick out integer numbers. The reserved word in this case is "NUMERO". If this is used in the syntax then ANALYSE processes integer numbers, i.e., a sequence of digits on one card, and puts this sequence as a character string in the parameter

STRING. INTERPRET may then decode this string of characters. The symbol number corresponding to NUMERO is passed to the Parse, and the corresponding character string is passed to INTERPRET when an integer is isolated as a left-most reducible substring. Other conventions of the Scan follow.

Comments. If it finds the word "COMMENT" used in the input stream, the Scan will skip symbols until it finds a semi-colon. Thus comments may be inserted between any pair of symbols in the input stream. Scan always sees COMMENT as a key-word. The next symbol may appear immediately after the semi-colon marking the end of the comment. Any symbols may be used between the word COMMENT and its closing semi-colon. This entire string does not exist as far as the Parse is concerned because it is ignored by the Scan.

Character Strings. A method is provided for handling character strings. This allows the user to incorporate them in the language he is designing. If the user includes the reserved word "ANYSTRING" in his syntax then ANALYSE will process strings delimited by quotes. The maximum length of string is 256 characters. Strings longer than this are truncated on the right without a warning being given. Any character may appear between the quotes but the quote. An example is 'any kind of string'. The symbol number for ANYSTRING is sent to the Parse and the character string between the quotes is passed to INTERPRET in the parameter STRING.

2.3.2 INTERPRET PARAMETERS

The parameter list and declarations required in the user written external procedure INTERPRET are as follows:

```
INTERPRET;
     PROCEDURE (PROD,LEFT,RIGHT,STACK,VARIABLE,SYMBOL,STRING,INFO,
     STATUS);
     DCL (PROD,LEFT,RIGHT,STACK(*),STATUS,INFO) FIXED BIN(15),
     SYMBOL(*) CHAR(12), VARIABLE(*) CHAR(12) VAR,
     STRING CHAR(256) VAR;
```

A description of the parameters and their use is given in the order of their appearance.

<u>PROD</u>. This is the current production number. When ANALYSE finds a left-most reducible substring it searches the list of Right Parts for a match and assigns that production number to PROD. The numbers used here are the same as those listed with the productions in the printout of both TESTPREC and EXTRACT. If ANALYSE isolates a string of symbols which does not match the right part of any production, then PROD is assigned the value 0.

<u>LEFT, RIGHT</u>. These are pointers in the parameter STACK. They point respectively to the left-most and right-most symbols of the left-most reducible substring isolated by ANALYSE. As the STACK is numbered from the bottom up, RIGHT is greater than or equal to LEFT. Using these two pointers the user can access any or all of the symbols in this isolated Right Part.

<u>STACK</u>. This is the symbol stack used by ANALYSE during the Parse. The parameter RIGHT indicates the number of symbols in the STACK.

The sequence from LEFT to RIGHT is the Right Part of the production PROD. The STACK values are the internal numbers used to represent the symbols in ANALYSE. Although the user has access to the entire STACK, only a small part of it should be used at any call of INTERPRET. INTERPRET should never modify the STACK.

VARIABLE. This parameter is only of use when VAR_TABLE appears in the syntax of the language. This array contains the character strings (of maximum length 12) representing identifiers in the Source program. The actual character strings used are of little importance internally because everything can be represented by numbers or addresses. This array then is provided in case the implementor wishes to write out error messages such as: THE VARIABLE _____ WAS NOT DECLARED. See the explanation of INFO.

SYMBOL. This array contains the character strings (of maximum length 12) representing all of the symbols (both Terminal and Nonterminal) used in the syntax of the language. These are the same symbols as those printed down the left side of the precedence matrix by the program TESTPREC. By using the internal symbol numbers substituted by ANALYSE as the subscripts to this array, the original symbols may be obtained.

STRING. This is a varying length character string with a maximum size of 256. It is used in three cases:

1. To pass characters of an integer number when NUMERO is used.
2. To pass a string of characters when ANYSTRING is used.
3. To pass the characters representing an identifier when VAR_TABLE or VAR_STRING is used.

INFO. Its sole purpose is to pass a subscript for the array VARIABLE when identifiers are handled by using VAR_TABLE.

STATUS. This allows INTERPRET to pass information to ANALYSE concerning the status of the present analysis. It has an initial value of 0 set by ANALYSE. There are three values which INTERPRET may assign to it:

1 - means to continue the lexical scan but no longer parse or call INTERPRET.

2 - means to continue both the Scan and Parse but no longer call INTERPRET.

3 - means to terminate execution immediately.

If STATUS is set to 1 or 2, a message telling what action is being taken is printed by ANALYSE enabling the user to see the point at which this occurred in the Source listing.

The following declaration may also be included in INTERPRET:

```
DCL SCAN_ERR BIT(1) EXTERNAL;
```

By testing to see if this is true, INTERPRET can determine if an error has been detected by the Scan in ANALYSE, and take some appropriate action.

2.3.3 IMPLEMENTATION AIDS

Value Stack. As the syntactic analysis proceeds it will usually be necessary for the user to retain information about the symbols in the STACK. This information may be memory addresses of identifiers, or parameters of loops etc. A convenient way of retaining this information is by constructing one or more VALUE stacks of the same size as the

STACK used during the Parse. In ANALYSE this is a one dimensional array of size 50. By putting the information in the Value stacks in the same corresponding positions as the symbols appear in the STACK, the user is able to extract this information later by using the parameters LEFT and RIGHT. All data areas which are to be retained from one call of INTERPRET to the next should be declared with the attribute STATIC.

Parameters Passed to ANALYSE through JCL. Some parameters may be passed to ANALYSE by means of the PARM list on the EXEC card. This enables the user to specify some options to ANALYSE in the JCL. The allowed parameters are:

1. COUNT(@) - In this case @ specifies some special character such as a statement delimiter which the user wants counted and listed as the statement number in the source listing.
2. OPT=1 - This indicates that only a lexical scan is to be made on the input string. (i.e., the string is not to be parsed and INTERPRET is not to be called)
3. OPT=2 - This indicates that ANALYSE is to perform a lexical scan and a syntactic analysis on the input string but INTERPRET is not to be called.
4. TRACE - This indicates that a trace of the parse is to be printed down the right-hand side of the source listing. Each time the symbol at the top of the stack is compared with the symbol coming in from the scan the following message is printed:

   XXX YYY @

XXX is the symbol at the top of the STACK, YYY is the incoming symbol from the Scan, and @ is the precedence relation between these two symbols.

If either OPT=1 or OPT=2 is specified a null external procedure called INTERPRET must still be provided. It may consist of something like this:

```
INTERPRET;
     PROCEDURE;
     RETURN;
     END;
```

Using this facility one can test one's language design by having ANALYSE perform a syntactic analysis on a source program written in the new language without providing semantic routines.

2.3.4 ANALYSE ERROR MESSAGES

ANALYSE detects errors in the Source program during both the Scan and the Parse.

Errors found during the Scan can be divided into two groups. In both of these cases the error is usually caused by the Scan isolating a symbol in the Source program which it is not equipped to handle. In the first case, an error message is printed and the offending symbol is deleted from the source. The Scan then isolates the next incoming symbol and presents this to the Parse. When the Scan recovers from an error in this way SCAN_ERR is set to true, informing INTERPRET that an error has occurred. In the second case ANALYSE finds itself in a situation from which it can not recover and therefore prints an error message and then terminates. A list of the error messages printed by ANALYSE is given for each case.

Warnings.

****** THE OPERATOR _ WAS USED BUT DOES NOT APPEAR IN THE PRODUCTIONS - DELETED

****** THE SYMBOL _ WAS FOUND BUT NEITHER THIS SYMBOL NOR NUMERO OCCURS IN THE SYNTAX

Terminal Errors.

****** - A VARIABLE _____ WAS FOUND BUT NEITHER VAR_STRING NOR VAR_TABLE APPEARS IN THE SYNTAX

****** - THE ANALYSER LIMIT ON THE NUMBER OF VARIABLES ALLOWED HAS BEEN EXCEEDED EITHER REDUCE THE # OF VARIABLE NAMES USED OR REPLACE VAR_TABLE BY VAR_STRING AND HANDLE THE VARIABLES IN INTERPRET

When the Parse isolates a left-most reducible substring which does not match the right-part of any production in the language, the following error message is printed:

****** - INVALID STACK SEQUENCE- XXX YYY ZZZ

In this example the three symbols XXX, YYY and ZZZ appear in this reducible substring. The number of symbols in this substring may be one or more. INTERPRET is then called with PROD set to 0.

The error message

****** - END OF FILE

is printed when the Scan tries to read another symbol after the end of the input string. Execution then terminates at this point.

The error message

****** - PARSE TERMINATED BY SYNTAX ANALYSER. SCAN CONTINUING

is printed when the parse finds the relationship ·> between the symbol at the top of the STACK and the incoming symbol, but the relationship <· does not occur between any of the symbols in the STACK.

Syntax Errors. A syntax error message as given by ANALYSE is of the following form:

```
*****SYNTAX*****          $
```

This means that the symbol with the $ printed under it in the Source listing has no precedence relationship with the symbol ANALYSE is comparing it to. This offending symbol is then deleted and the next incoming symbol compared. In this case SCAN_ERR is set to true.

If ANALYSE is performing both a Scan and a Parse of the input string (i.e., STATUS has a value of zero or two) then a syntax error means that this specified symbol has no precedence relationship with the symbol at the top of the STACK. Many syntax errors may be given in this case with each offending symbol being deleted until a symbol is found which has a relationship with the symbol at the top of the STACK. At this point the Parse continues.

If ANALYSE is only performing a lexical scan on the input (i.e., STATUS has a value of one, or OPT=1) then a syntax error means that the symbol cannot logically follow the one preceeding it in the source listing according to the syntax of the language. Again in this case offending symbols are deleted and the next incoming symbol is compared to the last valid symbol.

2.3.5 ERROR RECOVERY

As mentioned previously, when ANALYSE finds a Left-most Reducible substring which does not match the Right-part of any production, PROD is assigned a value of zero and then INTERPRET is called. When INTERPRET returns control, ANALYSE is unable to make the reduction in the normal way. If error recovery is not provided, the following error message is printed:

***** - NO ERROR RECOVERY PROVIDED - ONLY SCAN CONTINUING

If this occurs then the rest of the Source is not parsed. Although the Scan may find additional errors in the rest of the text, some errors may be missed on this run.

In order to allow ANALYSE to perform a more complete error check of the Source program a method of error recovery is provided. It would be more correct to call it Parse recovery than error recovery because it operates by deleting sections of the text in order to get the Parse started again. The normal sequence would probably be for the user to provide this error recovery and then set STATUS equal to 2 if PROD is passed to it with a value of zero. After this ANALYSE will recover if it can and will continue to parse the input but will no longer call INTERPRET. If STATUS is left alone, then INTERPRET is called again in the normal way after recovery is made.

As the user may not always want to provide error recovery, this facility is provided through the use of the EXTERNAL attribute. The following declaration is necessary:

```
DCL (FIX_UP(50,3) FIXED BIN(15),FIX_BIT(50,3) BIT(1),DEL BIT(1))
EXTERNAL;
```

By setting DEL to true the user informs ANALYSE that error recovery has been provided. This is accomplished by loading the two arrays FIX_UP and FIX_BIT. Using this information ANALYSE may delete part of the STACK and skip symbols using the Scan. The principle is that by deleting a section of code, the Parse is able to recover.

The array FIX_UP indicates the symbols ANALYSE is to look for in the STACK and in the incoming stream. The array FIX_BIT tells whether these symbols are to be kept or deleted when found. Fig. 12 shows the data loaded into FIX_UP and FIX_BIT to provide error recovery for the sample language. Fig. 13 shows the symbols corresponding to the internal symbol numbers in Fig. 12.

These arrays have 50 rows each with 3 columns. For any row I FIX_UP(I,1) indicates the symbol to be found in the STACK. If it is found then ANALYSE looks ahead in the incoming stream for the symbol FIX_UP(I,2). It is possible to list many symbols to look for in the incoming stream for a single symbol found in the STACK. This is accomplished by setting column 1 to zero for each row after row I until another STACK symbol is entered. Depending on the symbol found in the incoming stream, the user may decide he wants to delete deeper in the STACK. This is done by specifying the new symbol he wants to look for in the STACK in column 3 of the same row as the symbol which was found in the incoming stream. It is not necessary for the user to load this entire array. As there is always a non-zero number in column 2 for any valid row, ANALYSE looks for a zero in this column to indicate the end of the valid data. The numbers entered in this array are the internal symbol numbers which ANALYSE uses throughout. The user can get these from the printout of TESTPREC. Column 3 of FIX_UP must be set to zero for all valid rows of information where the user does not want ANALYSE to look deeper in the STACK. Information entered in the array FIX_BIT pertains to the same relative positions in FIX_UP. This must be loaded for every valid row of FIX_UP. Once ANALYSE has found the required symbols in the STACK and the incoming stream, it looks to FIX_BIT to see whether these symbols are to be kept or deleted. A true

FIX_UP

	1	2	3
1	10	10	0
2	0	7	0
3	5	10	0
4	0	7	0
5	21	10	0
6	0	26	0
7	51	51	0
8	0	10	0
9	14	29	0
10	0	31	0
11	0	40	0
12	0	42	0
13	0	10	10
14	0	0	0

FIX_BIT

1	2	3
0	1	0
1	0	0
0	1	0
1	1	0
1	0	0
0	1	0
0	1	0
1	0	0
0	1	0
0	1	0
0	1	0
0	1	0
0	0	1
0	0	0

Fig. 12

	1	2	3
1	;	;	
2		END	
3	BEGIN	;	
4		END	
5	IF-ELSE	;	
6		ELSE	
7	,	,	
8		;	
9	:=	+	
10		-	
11		*	
12		/	
13		;	;

Fig. 13

value (i.e., '1'B) means that the symbol is to be deleted, and a false value (i.e., '0'B) that the symbol is to be kept. If these arrays are carefully loaded then the Parse should be able to recover most of the time.

If the situation occurs where both the symbol in the STACK and the symbol in the incoming stream are to be deleted when found, then ANALYSE assumes that there is something special about this pair. This is useful in the case of something like Algol blocks. BEGIN and END could be listed as a "delete-delete" pair. When pairs like this occur, ANALYSE will never stop the search of the input string upon finding a symbol it is looking for, if more occurrences of the first of this pair than the second have been encountered during this search of the incoming stream. By including a pair like this in FIX_UP the user can prevent ANALYSE trying to start the Parse after deleting the BEGIN and then failing again when it discovers the END.

When ANALYSE starts the search of the STACK it begins with the symbol immediately before position LEFT, and works down from there. If none of the symbols it is looking for are found in the STACK then the following error message is printed:

****** - ERROR RECOVERY FAILED - ONLY SCAN CONTINUING

ANALYSE shows the section of the input stream which was deleted during error recovery through the use of the following messages:

SCAN DELETED FROM HERE—>

<—TO HERE

These messages may occur on the same line in the source listing or many lines apart depending on the amount of text which was deleted. The arrow heads mark this portion exactly.

Although this text which is skipped is not parsed, it is still given a thorough check by the Scan. Any error message which the Scan normally gives will be given in this case. This includes the SYNTAX error. In this case the symbols are not compared with the top of the STACK but with the most recent valid symbol before it in the incoming stream. See Fig. 15 for an example of the operation of error recovery.

2.3.6 WARNINGS

If the user incorporates any of the four reserved words (NUMERO, VAR_STRING, VAR_TABLE or ANYSTRING) in his language, it would be advantageous to have these symbols appear alone on the Right-side of productions. As the INTERPRET parameter STRING can only hold one item at a time, the user must make sure that his language does not allow any of these symbols or nonterminals directly reduced from them to occur adjacent to each other.

The parameter STRING is loaded by the Scan, so at any time the information in STRING may refer to the next incoming symbol rather than to a symbol in the STACK.

Even if the Scan deletes an incoming symbol because of some error, the parameter STRING may have already been changed. In this case the parse may continue normally after the symbol is deleted, but the information in STRING may be for the symbol deleted rather than a symbol in the STACK. The user should check SCAN_ERR in INTERPRET, and if it is true, should assume that STRING is in error. The same holds true of the parameter INFO.

ANALYSE inserts the relation $<\cdot$ before the first symbol in the source program and $\cdot>$ after the last to allow the program to parse to completion. The user must make sure that the first symbol in the program and therefore the first symbol in the STACK is not included in a reduction before the program parses to completion as there will then be no relation between the symbol introduced because of this reduction and the bottom of the STACK. Violating this will result in the Parse being terminated by ANALYSE.

Fig. 14 shows a procedure INTERPRET for the sample language. It does not provide semantics but serves to provide error recovery as shown in Fig. 13 to demonstrate this facility. A sample program written in this language is shown in Fig. 15. Fig. 16 shows the output of EXTRACT for a second sample language. Fig. 17 gives the procedure INTERPRET which is really an interpreter because it executes the instructions immediately instead of generating object code. Fig. 18 shows a sample program written in this language.

INTERPRET:

```
1        INTERPRET:
           PROCEDURE (PROD,LEFT,RIGHT,STACK,VARIABLE,SYMBOL,STRING,INFO,
           STATUS);
2          DCL (PROD,LEFT,RIGHT,STACK(*),STATUS,INFO) FIXED BIN(15),
           SYMBOL(*) CHAR(12),VARIABLE(*) CHAR(12) VAR,
           STRING CHAR(256) VAR;
3          DCL (FIX_UP(50,3) FIXED BIN(15),FIX_BIT(50,3) BIT(1),DEL BIT(1))
           EXTERNAL,ERR BIT(1) STATIC INITIAL ('0'B);
4          DCL SYSIN1 FILE STREAM INPUT;
        /* THE FOLLOWING SECTION OF CODE IS ONLY EXECUTED ON THE FIRST
           CALL OF 'ANALYSE'                                                 */
5          IF ERR THEN GO TO S2;
        /* 'SYSIN1' IS A CARD FILE WITH ERROR RECOVERY INFORMATION           */
7          ON ENDFILE(SYSIN1) GO TO S1;
9          OPEN FILE(SYSIN1);
10         DO I=1 TO 50;
11         GET FILE(SYSIN1) EDIT ((FIX_UP(I,J) DO J=1 TO 3))
           (X(2),F(3,0),X(1),F(3,0),X(1),F(3,0));
12         GET FILE(SYSIN1) EDIT ((FIX_BIT(I,J) DO J=1 TO 3))
           (X(2),B(1),X(1),B(1),X(1),B(1));
13         END;

14       S1: CLOSE FILE(SYSIN1);
15         ERR,DEL='1'B;
16       S2: IF PROD = 1 THEN DO;
18            PUT EDIT ('THE PROGRAM PARSED TO COMPLETION') (A) SKIP;
19            STATUS=3;
20         END;
21         RETURN;
22         END INTERPRET;
```

Fig. 14

```
ISN                                                      SOURCE LISTING

  1                    START
  1                    COMMENT     THIS PROGRAM IS DESIGNED TO DEMONSTRATE ERROR RECOVERY AND
  1                                TO SHOW SOME OF THE ERROR MESSAGES GIVEN BY 'ANALYSE';
  1                    BEGIN
  1                         INTEGER N,A,B,C,D,F,X,Y,TOT,AVG;
  2                         LABEL ST,STR;
  3                         COMMENT     THERE ARE 2 ERRORS IN THE FOLLOWING LINE:
  3                                     1. MISSING SEMI-COLON BEFORE A
  3                                     2. MISSING : AFTER D;
  3                         Y:=0     A:=0;   B:=0;   C:=0;   D =0;   F:=0;   TOT:=0;
****SYNTAX*****                    $
****SYNTAX*****                     $
****SYNTAX*****                       $

**** - INVALID STACK SEQUENCE-     STATLIST      ;              FACTOR
                                      SCAN DELETED FROM HERE-->
                                                                  <--TO HERE
**** - DELETED FROM STACK-
  9                         READ N;
 10                         TOT:=TOT+X;
 11                    ST:  READ X;
 12                              IF X > 75 THEN
 12                                   BEGIN
 12                                        A:=A+1;
 13                                        GO TO STR
 13                                   END
 13                              ELSE
 13                              COMMENT     THE UNMATCHED ')' CAUSED THE FOLLOWING ERROR;
 13                              IF X > 66) THEN

**** - INVALID STACK SEQUENCE-     EXPR          )
                SCAN DELETED FROM HERE-->
 13                                   BEGIN
 13                                        B:=B+1;
 14                                        GO TO STR
 14                                   END
 14                              ELSE
                                     <--TO HERE
**** - DELETED FROM STACK-      IF             CHOICE       >
 14                              IF X > 59 THEN
 14                                        C:=C+1
 14                              ELSE
 14                              IF X > 49 THEN
 14                                        D:=D+1;
 15                              COMMENT     THE SYNTAX ERROR WAS CAUSED BY THE SEMI-COLON
 15                                          BEFORE 'ELSE';
 15                              ELSE
****SYNTAX*****                  $
 15                                   F:=F+1;
 16                    STR: Y:=Y+1;
 17                             IF Y < N THEN GO TO ST;
 18                    WRITE N;
 19                    WRITE A,B,C,D,F;
 20
 20
 20                         COMMENT     THE PARSE HANGS AND RECOVERS TWICE IN THE FOLLOWING STATEMENT
```

Fig. 15

```
  20                          TOTAL:= (A+B-C)*(SUM/REST)+TERM)-33*I4/(AVG-66;

***** - INVALID STACK SEQUENCE-     EXPR          )
                                     SCAN DELETED FROM HERE-->
                                                              <--TO HERE
***** - DELETED FROM STACK-

***** - INVALID STACK SEQUENCE-     (             EXPR
                                                    SCAN DELETED FROM HERE-->
                                                                           <--TO HERE
***** - DELETED FROM STACK-     ;            VAR            :=            TERM-        /
  21                    AVG:=TOT/N;
  22                    WRITE AVG;
  23                    COMMENT    THE FOLLOWING TWO STATEMENTS HAVE AN '=' INSTEAT OF ':=';
  23                    ALPHABET='ABCDEFGHIJKLMNOPQRSTUVWXYZ';

***** - INVALID STACK SEQUENCE-     STATLIST     ;            FACTOR
        SCAN DELETED FROM HERE-->
                                                              <--TO HERE
***** - DELETED FROM STACK-
  24                    COMMENT 'ANALYSE' COULD NOT RECOVER AGAIN BECAUSE NONE OF THE SYMBOLS
  24                            IN THE FIRST COLUMN OF 'FIX_UP' REMAINED IN THE 'STACK';
  24                    DIGITS='0123456789';

***** - INVALID STACK SEQUENCE-     BEGIN         FACTOR
      SCAN DELETED FROM HERE-->

***** - ERROR RECOVERY FAILED - ONLY SCAN CONTINUING
  25                    X := A ? B;
***** THE OPERATOR ? WAS USED BUT DOES NOT APPEAR IN THE PRODUCTIONS. - DELETED
*****SYNTAX*****                $
  26                    AVG:=DIGITS;
  27                    WRITE AVG
  27                    END
  27                    FINISH

***** - END OF FILE
```

Fig. 15 cont'd

PRODUCTIONS

1	PGRAM	::=	START	BODY	FINISH
2	BODY	::=	STATLIST		
3	STATLIST	::=	STATLIST	;	STATMENT
4		::=	STATMENT		
5	STATMENT	::=	VAR	=	EXPR
6		::=	WRITE	VAR	
7	EXPR	::=	EXPR-		
8	EXPR-	::=	EXPR-	+	TERM
9		::=	EXPR-	-	TERM
10		::=	+	TERM	
11		::=	-	TERM	
12		::=	TERM		
13	TERM	::=	TERM-		
14	TERM-	::=	TERM-	*	FACTOR
15		::=	TERM-	/	FACTOR
16		::=	FACTOR		
17	FACTOR	::=	(	EXPR	)
18		::=	INTEGER		
19		::=	REAL		
20		::=	VAR		
21	VAR	::=	VAR_TABLE		
22	INTEGER	::=	NUMERC		
23	FRACTION	::=	.	NUMERC	
24	REAL	::=	INTEGER	FRACTION	
25		::=	FRACTION		

ABLES LOADED SUCCESSFULLY

Fig. 16

INTERPRET:

```
1      INTERPRET:
          PROCEDURE (PROD,LEFT,RIGHT,STACK,VARIABLE,SYMBOL,STRING,INFO,
          STATUS);

          /*             DECLARATION OF 'INTERPRET' PARAMETERS                  */
2         DCL (PROD,LEFT,RIGHT,STACK(*),STATUS,INFO) FIXED BIN(15),
          SYMBOL(*) CHAR(12),VARIABLE(*) CHAR(12) VAR,
          STRING CHAR(256) VAR;

          /*                  'INTERPRET' DATA AREAS
          MEMORY   - STORES THE CURRENT VALUE OF ALL IDENTIFIERS
                   - POSITIONS CORRESPOND TO 'VARIABLE'
          VALUE    - HOLDS CURRENT VALUES OF IDENTIFIERS AND NUMBERS
                   - POSITIONS CORRESPOND TO 'STACK'
          VALUE_1  - HOLDS 'INFO' FOR IDENTIFIERS
                   - POSITIONS CORRESPOND TO 'STACK'
          LABEL    - GIVES LABEL TO JUMP TO FOR EACH VALUE OF 'PROD'
          NBR      - CHARACTER REPRESENTATION OF ALL DIGITS
                                                                                */
3         DCL ((MEMORY(250),VALUE(50),K) BIN FLOAT(31),
          (VALUE_1(50),LGTH,J,I) BIN FIXED(15)) STATIC,Y CHAR(1);
4         DCL LABEL(0:25) LABEL INITIAL(L0,L1,L_END,L_END,L_END,L5,
          L6,L_END,L8,L9,L10,L11,L_END,L_END,L14,L15,L_END,L17,L_END,
          L_END,L_END,L21,L22,L23,L24,L_END);
5         DCL NBR(0:9) STATIC CHAR(1) INITIAL('0','1','2','3','4','5',
          '6','7','8','9');
6         GO TO LABEL(PROD);

7      L0:     /* LEFT-MOST REDUCIBLE SUBSTRING DOES NOT MATCH ANY PROD.      */
          STATUS=3;
8         RETURN;

9      L1:     /* PGRAM      ::=  START       BODY         FINISH             */
          STATUS=3;
10        PUT EDIT ('THE PROGRAM PARSED TO COMPLETION') (A) SKIP;
11        RETURN;

12     L5:     /* STATMENT   ::=  VAR         =            EXPR               */
          MEMORY(VALUE_1(LEFT))=VALUE(RIGHT);
13        RETURN;

14     L6:     /* STATMENT   ::=  WRITE       VAR                             */
          PUT EDIT ('ANSWER ',VALUE(RIGHT)) (A,F(12,5)) SKIP;
15        RETURN;

16     L8:     /* EXPR-           EXPR-       +            TERM               */
          VALUE(LEFT)=VALUE(LEFT)+VALUE(RIGHT);
17        RETURN;

18     L9:     /* EXPR-      ::=  EXPR-       -            TERM               */
          VALUE(LEFT)=VALUE(LEFT)-VALUE(RIGHT);
19        RETURN;

20     L10:    /* EXPR-      ::=  +           TERM                            */
          VALUE(LEFT)=VALUE(RIGHT);
21        RETURN;
```

Fig. 17

```
INTERPRET:

22          L11:   /* EXPR-        ::=  -            TERM                                */
               VALUE(LEFT)=-VALUE(RIGHT);
23             RETURN;

24          L14:   /* TERM-        ::=  TERM-        *            FACTOR                 */
               VALUE(LEFT)=VALUE(LEFT)*VALUE(RIGHT);
25             RETURN;

26          L15:   /* TERM-        ::=  TERM-        /            FACTOR                 */
               VALUE(LEFT)=VALUE(LEFT)/VALUE(RIGHT);
27             RETURN;

28          L17:   /* FACTOR       ::=  (            EXPR         )                      */
               VALUE(LEFT)=VALUE(LEFT+1);
29             RETURN;

30          L21:   /* VAR          ::=  VAR_TABLE                                        */
               VALUE_1(LEFT)=INFO;
31             VALUE(LEFT)=MEMORY(INFO);
32             RETURN;

33          L22:   /* INTEGER      ::=  NUMERO                                           */
               VALUE(LEFT)=0;
34             LGTH=LENGTH(STRING);
35             DO I=1 TO LGTH;
36                Y=SUBSTR(STRING,I,1);
37                DO J=0 TO 9;
38                    IF Y=NBR(J) THEN GO TO C3;
40                END;
41              C3: VALUE(LEFT)=VALUE(LEFT)*10+J;
42             END;
43             RETURN;

44          L23:   /* FRACTION     ::=  .            NUMERO                              */
               VALUE(LEFT)=0;
45             LGTH=LENGTH(STRING);
46             DO I=1 TO LGTH;
47                Y=SUBSTR(STRING,LGTH+1-I,1);
48                DO J=0 TO 9;
49                   IF Y=NBR(J) THEN GO TO C4;
51                END;
52              C4: VALUE(LEFT)=VALUE(LEFT)*.1+J;
53             END;
54             VALUE(LEFT)=VALUE(LEFT)*.1;
55             RETURN;

56          L24:   /* REAL         ::=  INTEGER     FRACTION                             */
               VALUE(LEFT)=VALUE(LEFT)+VALUE(RIGHT);

57          L_END: /* NULL INTERPRETATION RULE                                           */
               RETURN;
58             END INTERPRET;
```

Fig. 17 cont'd

```
ISN                                        SOURCE LISTING

  1                     START
  1                     COMMENT        THIS IS A VERY SIMPLE PROGRAM TO DEMONSTRATE THAT
  1                                    THE PROCEDURE 'INTERPRET' DOES WORK;
  1                     A1 = 434.352;
  2                     A2 = 963.617;
  3                     A3 = 1000.961;
  4                     A4 = 1.54369;
  5                     A5 = 3.1461;
  6                     SUM=A1+A2+A3+A4+A5;
  7                     WRITE SUM;
SWER    2403.61979
  8                     AVERAGE = SUM/5;
  9                     WRITE AVERAGE;
SWER     480.72396
 10                     TEST=1.1;
 11                     WRITE TEST;
SWER       1.10000
 12                     TEST1=0001.002;
 13                     WRITE TEST1;
SWER       1.00200
 14                     FRACTION=.00035;
 15                     WRITE FRACTION;
SWER       0.00035
 16                     SIXTEENSQUARED=(4+3+9)*(20-6+2)*(31-15)/(3*5+1);
 17                     WRITE SIXTEENSQUARED
 17                     FINISH
SWER     256.00000
E PROGRAM PARSED TO COMPLETION
```

Fig. 18

CHAPTER 3

PROGRAM DESCRIPTIONS

A brief description is given of each of the three programs of the System. These discussions are not intended to be complete descriptions of the workings of the programs. Only those points which were considered to be of vital importance or difficult to understand are covered. Complete program listings can be found in the appendices.

3.1 TESTPREC

This program reads in the data cards (i.e., the productions of the grammar) one at a time, into the string TEST and then a blank is concatenated onto the right. Column 1 is checked to see if a new left-part is being defined and then all symbols delimited by blanks are picked out of this string. As these symbols are found they are loaded into the array INPUT, with INPUT(1) being the left-part. All symbols are checked in the array SYMBOL to see if they have already been encountered, and if not are entered. The array TERM is used to note the symbols which occurred on the left-side of productions.

The limit of 180 symbols in the syntax which is imposed by TESTPREC is necessary because of a PL/1 restriction on the size of BIT STRINGS. It was found to be much more efficient for TESTPREC to use bit strings to store certain information, and do its own addressing than to use arrays with the attribute BIT(1). The three bit strings used by TESTPREC are LEFT, RIGHT and EQUAL. The numbers used to represent symbols in accessing locations in these strings and later in the precedence matrix, are the symbol locations in the array SYMBOL. These strings are of size 32400 which is 180 X 180. For two symbol numbers I and J the expression used to access the corresponding bit is 180*(I-1)+J. A true value has the following significance in the different strings:

LEFT - means that symbol J is the left-most symbol on the right-hand side of a production in which symbol I is the left-part.

RIGHT - means that symbol J is the right-most symbol on the right-hand side of a production in which symbol I is the left-part.

EQUAL - means that symbol J occurs to the right of and adjacent to symbol I on the right-hand side of a production.

The pertinent entries are made in these strings for each production as it is encountered. This process continues until the entire grammar has been read in.

The precedence matrix is declared to be a CHARACTER STRING of size N X N where N is the total number of symbols in the syntax. Addressing is the same as for bit strings. The arrays NONT and TERMINAL are declared and then loaded with the Nonterminal and Terminal symbols respectively. These are then printed out under the corresponding headings.

To be useful in establishing the precedence relationships between symbols, the strings LEFT and RIGHT have to be changed to show all of the left-most and right-most symbols derivable respectively. Warshall's algorithm (3) was used to perform this transformation on the strings LEFT and RIGHT.

It was considered best to use a character string (PREC) for the precedence matrix because it could then be printed out once it had been loaded without any conversion. This string is first initialized to blanks. Loading the precedence matrix consists of inserting the characters $\doteq$, $<$ and $>$ in the correct places in this string.

The string EQUAL is searched with J having values from 1 to N for all I from 1 to N. Whenever a true bit is found, the symbol "$\doteq$" is inserted in the corresponding position in PREC. If J is a nonterminal then the symbol "$<$" is entered in PREC between symbol I and every left derivable symbol J (as determined from LEFT). Next I is tested to see if it is a nonterminal. If it is, then the symbol "$>$" is entered in PREC between every right derivable symbol of I (as determined from RIGHT) and the symbol J. Finally if both I and J are nonterminals then entries of "$>$" are made between every right derivable symbol of I and left derivable symbol of J.

As can be seen PREC was loaded by working through the definitions of the precedence relations as given by Wirth and Weber (1). If a relation was to be entered in PREC and another relation was already present in that location then a precedence violation exists.

To make the definitions more meaningful, the precedence violations are explained in the error messages in terms of the definitions. The error messages consist of a printout of the two symbols between which there is the violation, and the two relations which were found (the one already in PREC followed by the one which was to be inserted). Explanations are then printed to show how the relations were derived according to the definitions.

It is an easy matter to explain the current relation to be inserted because all of the pertinent information is still available. To give an explanation of the relation which was already in the precedence matrix is more involved. If this relation was "=" then it is easy to give a description because the symbols obviously occur adjacent to each other on the right side of some production. If the relation was "<" or ">" then the procedure LESS or GREAT respectively is called.

LESS. The input to this procedure consists of the two symbol numbers (called X and Y) giving the location in PREC of the relation "<". LESS begins a search in the string LEFT to find the symbols for which Y is a left derivable symbol. When it finds one it checks in the string EQUAL to see if symbol X has equal precedence with this symbol. If this is the case then an error message is printed describing how this relationship was derived.

GREAT. The inputs for this procedure are the same as those for LESS. As there are two possibilities for the derivation of the relation ">" the first case is checked completely first. GREAT begins a search in the string RIGHT to find the symbols for which X is a right derivable symbol. When it finds one it checks in the string EQUAL to see if this symbol has equal precedence with Y. If this is the case then an error message is printed. If this search is unsuccessful in finding the cause of this relation then a second search is begun. Once again the string RIGHT is searched to find the symbols for which X is a right derivable symbol. When one is found, a search is made of LEFT to find the symbols for which Y is a left derivable symbol. When one is found here, the string EQUAL is checked to see if these two symbols which have been found are of equal precedence. If this is the case then the error message is printed. The procedures LESS and GREAT are always successful in their searches because if a relation is entered in the precedence matrix, its cause will be found by working through the definitions.

When an error message states that one symbol has equal precedence with another, it also states the production in which they appear adjacent to each other. This information is stored in the array LINE as the productions are being read in at the beginning of the program. The array LINE is of size 400 X 2. (i.e., LINE(400,2)) Whenever an entry is made in the string EQUAL an entry is also made in LINE. LINE(I,1) where I is any number from 1 to 400, is loaded with the address of the current relation being inserted in EQUAL. (for example: 180*(L-1)+K where L and K are symbol numbers) LINE(I,2) is then loaded with the current production line number.

It is the job of the procedure LINE_NO to return the line number when given the address in the precedence matrix of the relation "=". LINE_NO searches LINE(I,1) for all I until it finds one that matches the address given. It then returns LINE_NO(I,2) for that I, thus giving the production in which the two symbols occurred adjacent to each other.

The method used to calculate the F and G functions is one described by Wirth (4). This was found to be reasonably fast and not expensive of core storage. One of its strong points is that it can recognize quickly if F and G Functions do not exist.

The standard output of this program is a list of the productions, the terminal and nonterminal symbols and any error messages which were generated. The precedence matrix and F and G Functions are printed if MATRIX and FUNCTIONS respectively are specified in the PARM list on the EXEC card. This PARM list is a PL/1 facility which allows a varying length character string (PARM) of maximum length 100 to be passed to the program at run time.

The final version of TESTPREC is the result of much experimentation with different techniques. A method using Boolean Matrices to determine precedence relations as developed by Martin (5) was tried, but it was found to be very expensive in core storage and to be very slow. In this test two-dimensional arrays with the attribute BIT(1) were used. If bit strings were used instead the execution time would probably show some improvement, but the storage needed would still remain high. The method developed by Floyd (2) for finding F and G Functions was tried. Its greatest fault is the time it takes to determine that no F and G Functions exist for a particular grammar. A method developed by Bell (6) for finding F and G Functions was tried but it requires much more core storage than Wirth's method which was finally used. In addition it was not considered very useful to give an "almost F" and an "almost G" instead of saying that no precedence functions exist.

A free form of input was provided for TESTPREC in order to make it easier to use through a typewriter terminal or CRT.

The program was considerably lengthened by the inclusion of the detailed error messages. These error messages do not cause an increase in execution time for a correct grammar. In the case of a grammar which is not syntactically correct it was considered worth the extra computer time for the saving in human time to track down the errors. The program listing is given in Appendix A.

3.2 EXTRACT

The purpose of this program is to read in the productions of a simple precedence grammar and to create a file called TABLES with the attribute FILE STREAM OUTPUT into which is put all the pertinent information about the grammar which ANALYSE needs to perform a lexical scan and syntactic analysis on a source program written in this language

It would have been possible to make TESTPREC perform this task as well as its present task, but this would have made it much longer and would have put a heavy overhead on every run which was not successful. On the assumption that TESTPREC may have to be run many times during language development and TABLES only has to be loaded once after a successful run, it was decided to have a separate program for this purpose.

Because of the similarity in purpose of these two programs, code was taken directly from TESTPREC wherever possible in the development of EXTRACT. This includes reading in the productions in free format and building up the array SYMBOL and strings LEFT, RIGHT and EQUAL; the conversion of LEFT and RIGHT using Warshall's theorem, and the building up of the precedence matrix PREC. The detailed error messages given by TESTPREC were not included in this program, because the grammar is supposed to be syntactically correct before it is run using EXTRACT. For this reason the program terminates immediately if it finds a precedence violation.

All of the identifiers, arrays etc. which are loaded into TABLES by EXTRACT and later read in by ANALYSE have the same names in both programs. A description is given of all the information put into TABLES in the order in which it is loaded: (all dimensions given are those declared in ANALYSE)

N - This is the total number of different symbols occurring in the grammar.

LOC - This is the total number of symbols (counting all occurrences) appearing on the right-hand sides of all the productions.

M - This is the total number of productions in the grammar.

SYMBOL - This is a one-dimensional array of size N with the attribute CHAR(12) containing all of the symbols in the syntax in the order of their occurrence.

NUMB - This is a one-dimensional array of size LOC containing internal symbol numbers for all symbols occurring on the right-hand side of productions.

PROD - This is a two-dimensional array declared PROD(M,3). There is one row in PROD for each production in the grammar. For any I less than M, PROD(I,1) points to the next row in PROD which refers to a production of the same length. (i.e., the same number of symbols on the right-hand side) Prod(I,2) points to the location in the array NUMB where the right-hand side of production I is listed. PROD(I,3) is the internal symbol number of the symbol on the left side of production I.

LGTH_POINTER - This is a one-dimensional array of size 6. For I from 1 to 6, LGTH_POINTER(I) points to the first row in array PROD referring to a production of length I.

ANYSTRING, VAR_STRING, VAR_TABLE, NUMERO - These are all reserved words in the syntax. They are initially set to zero but if they appear in the syntax then they have the value of their own internal symbol numbers.

I1 - This is the total number of terminal symbols which begin with a letter followed by a sequence of letters or digits.

I2 - This is the total number of terminal symbols which consist of a single special character.

ID - This is the total number of terminal symbols which consist of two special characters.

KEY_WORD - This is a one-dimensional array of size I1 with the attribute CHAR(12). It contains the character strings for all terminal symbols which begin with a letter followed by a sequence of letters or digits. These are loaded in order from shortest to longest.

KEY_WORD_NO - This is a one-dimensional array of size I1. It contains the internal symbol numbers for all of the entries in KEY_WORD.

POINTER - This is a one-dimensional array of size 13. For any I from 1 to 12, POINTER(I) gives the location in KEY_WORD of the first entry of length I.

OPERATOR - This is a one-dimensional array of size I2 with the attribute CHAR(1). It contains the character representations of all the single special characters.

OPERATOR_NO - This is a one-dimensional array of size I2. It contains the internal symbol numbers for all of the entries in OPERATOR.

DOUBLE_OP - This is a one-dimensional array of size ID with the attribute CHAR(2). It contains the character representations of all the terminal symbols consisting of two special characters.

DOUBLE_OP_NO - This is a one-dimensional array of size ID. It contains the internal symbol numbers for all of the entries in DOUBLE_OP.

PREC - This is a character string of size N X N. In it is stored the entire precedence matrix.

If for some reason the tables are not loaded successfully on some run of EXTRACT, the user must make sure that the file TABLES is deleted before the next run. Failure to do this will result in EXTRACT trying to create a file which already exists and will cause a JCL error. The program listing is given in Appendix B.

3.3 ANALYSE

This program is designed to read in the file TABLES which was created by EXTRACT and contains information about a particular simple precedence grammar, and then perform a lexical scan and a syntactic analysis on a source program written in this language. The user written semantic routine INTERPRET is called each time a reduction is to be made.

The program begins execution by reading in the information in TABLES and creating and initializing data areas. A description of the information contained in TABLES is given in the write-up of EXTRACT.

The source program is read in one card at a time into the string IN from the SYSIN data set. The card image is put in columns 1 to 80 with column 81 being set to a blank. The blank acts as a delimiter between symbols and allows various portions of the scan to search for the end of a terminal symbol without having to check continuously if they have run over the end of the string IN. Although this mechanism saves much time during the scan it prevents the user from splitting terminal symbols over the end of cards in the source programs.

A short description of the logic of the Scan accompanied by a flow chart is given in the User's Manual. (Chapter 2)

The program listing is in Appendix C.

The array VARIABLE is used to store the identifiers as they are encountered in the source program when VAR_TABLE is specified in the syntax. When an identifier is found this array is searched to see if the identifier has already been entered and if not, a new entry is made. Each time a search is made, only those entries which are the same length (from 1 to 12 characters) are compared. Using the length (less than or equal to 12) of the new identifier as the subscript of array VAR_PT_1 gives the location in VARIABLE of the first entry of that length. If a match is not found here, then using the current subscript of VARIABLE as the subscript of VARIABLE_1 gives the next position in VARIABLE with an identifier this length. A value of zero returned by VAR_PT_1 or VARIABLE_1 indicates that the search has failed. The array VAR_PT_2 is used to indicate the positions in VARIABLE of the last entries of each length. This enables the updating of VARIABLE_1 when new identifiers are chained on to the end.

As the symbols at the top of the STACK are compared with the incoming symbols from the Scan during parsing, all occurrences of the relation "<" are stored in the array SAVE. When the relation ">" is found, a check is made of the last entry in SAVE to find the corresponding relation "<" and the left-most reducible substring is then isolated.

Once this substring has been found it is necessary to determine which production it represents. First the number of symbols in this substring is used as the subscript in the array LGTH_POINTER. This indicates the first row in the array PROD referring to productions of this length. If the value returned by LGTH_POINTER is I where I has a value from 1 to M (the number of productions) then PROD(I,2) gives the location in the array NUMB where the internal symbol numbers for the right-hand side of production I are stored. Knowing the number of symbols in the substring isolated by the parse enables a comparison to be made between the entries in NUMB and the symbols in this substring. If the comparison does not show that all of these symbols are the same, which would indicate that I is the correct production, then PROD(I,1) gives the next row in PROD which refers to a production of that same length. If these comparisons indicate that the substring found by the parse is the right side of some production I then I is passed to INTERPRET as the production number. If there are more than 6 symbols in the substring found by the parse, or either LGTH_POINTER or PROD(I,1) gives a value of zero, then the substring does not match any of the productions of the language and a production number of zero is sent to INTERPRET.

When control is returned to ANALYSE by INTERPRET the substring in the STACK is replaced by PROD(I,3) which is the left-side of production I.

When control is returned after passing a production number of zero, DEL is checked to see if error recovery has been provided. If DEL is false then ANALYSE only scans the remaining portion of the source program. When DEL is true and error recovery is being attempted for the first time, a special section of coding is executed to gather information from the arrays FIX_UP and FIX_BIT which helps to speed up the error recovery on this and all succeeding attempts.

A detailed description of error recovery and the arrays FIX_UP and FIX_BIT from the user's point of view was given in Chapter 2.

Two arrays are loaded by this special section. The first NO_POINTER contains pointers to the array FIX_UP indicating the rows in which the first column is not zero. When fully loaded, NO_POINTER indicates all of the rows in FIX_UP with non-zero first columns and thus all of the symbols which ANALYSE is to look for in the STACK. All occurrences of "delete-delete" pairs (i.e., cases where both the symbol in the STACK and the one in the incoming stream are to be deleted when found) are stored in the array DEL_PRS with DEL_PRS(I,1) being the STACK symbol and DEL_PRS(I,2) being the incoming symbol for the I'th "delete-delete" pair. Once these arrays are loaded an attempt is made at error recovery.

Each symbol in the STACK starting with the symbol immediately before the left-most reducible substring which ANALYSE last isolated, is compared with all of the symbols indicated by NO_POINTER in FIX_UP. Once a match has been found symbols are flushed from the Scan until a match is found with one of the symbols given in column two of FIX_UP from the row given by NO_POINTER(I) (where I is the entry for the symbol found in the STACK) to NO_POINTER(I+1)-1. While these symbols are being flushed all occurrences of any of the symbols in the "delete-delete" pairs are noted. When a correct symbol is found a check is made to see if there have been more occurrences of the first symbol of any of the "delete-delete" pairs than the second. If this is the case, then symbols are flushed from the Scan until another match is found. Once a match is made and the "delete-delete" pair requirement is met, column 3 of the same row of FIX_UP is checked to see if there is a non-zero entry. If there is then this indicates that additional symbols are to be deleted from the STACK until this symbol is found. Once the desired symbols in both the STACK and the Scan have been found, FIX_BIT is checked to see whether these symbols are to be kept or deleted. A true value indicates delete and false keep. Once this is done the parse continues.

CHAPTER 4

CONCLUSIONS

The System described in this paper has been used by several groups of graduate students at the University of Manitoba over a period of a few months in the design of a language and implementation of a compiler for it as a term project.

During this trial period no insurmountable problems were encountered and no program bugs were found.

It would be possible to extend the idea of a System for language development to include a larger set of languages than the present System allows, and this would greatly add to the versatility of the idea.

The present System, although restricted in that it can only handle Simple Precedence Languages, is a very useful tool and demonstrates some of the advantages to be gained by this approach to language development.

APPENDIX A

LISTING OF TESTPREC

TESTPREC:

```
1        TESTPREC:
            PROCEDURE (PARM) OPTIONS(MAIN);
2           DCL PARM CHAR(100) VAR;
3           DCL (INPUT(15),SYMBOL(180),X) CHAR(12),COMMA CHAR(1);
4           DCL (LEFT,RIGHT,EQUAL) BIT(32400),TERM(180) BIT(1),
            (A,B) CHAR(1),(I,J,K,L,N,M,NON,NUM,EQ,Z,Y,LINE(400,2),LGTH,LGTH1)
            FIXED BIN(15),(ERR,ERR1,PREC_MAT,FIND_F_G) BIT (1);
5           DCL TEST CHAR(81) VAR,ELEMENT CHAR(80) VAR, L2 FIXED BIN(15);
6           DCL LINE_NO ENTRY (FIXED BIN(15),FIXED BIN(15));
7           OPEN FILE(SYSPRINT) PRINT LINESIZE(132) PAGESIZE(61);
8           ON ENDFILE (SYSIN) GO TO MATRIX;
            /*           MAJOR DATA AREAS
                  SYMBOL - STORES THE SYMBOLS USED IN THE SYNTAX
                  EQUAL  - NOTES SYMBOLS WHICH OCCUR ADJACENT TO EACH OTHER ON
                           THE RIGHT-HAND SIDE OF PRODUCTIONS
                  LEFT   - NOTES LEFT-MOST SYMBOLS ON THE
                           RIGHT-HAND SIDE OF PRODUCTIONS
                  RIGHT  - NOTES RIGHT-MOST SYMBOLS ON THE
                           RIGHT-HAND SIDE OF PRODUCTIONS
                  TERM   - NOTES WHICH SYMBOLS ARE NONTERMINAL              */
10          COMMA=',';
11          EQ=0;
12          FIND_F_G,PREC_MAT='1'B;
13          ERR,ERR1='0'B;
14          NUM=180;
15          NON=0;
16          B=' ';
17          INPUT(7)=B;
18          LEFT,RIGHT,EQUAL='0'B;
19          TERM='1'B;
20          M=0;
21          N=0;
            /*     CHECK 'PARM' LIST TO SEE IF OPTIONAL OUTPUT REQUESTED    */
22          LGTH=LENGTH(PARM);
23          IF LGTH=0 THEN GO TO START;
25          IF INDEX(PARM,'FUNCTIONS') > 0 THEN FIND_F_G='0'B;
27          IF INDEX(PARM,'MATRIX') > 0 THEN PREC_MAT='0'B;

            /*     READ IN PRODUCTIONS IN FREE FORMAT AND
                   LOAD MAJOR DATA AREAS                                    */
29       START:
            PUT EDIT ('PRODUCTIONS') (X(54),A);
30          GET EDIT (TEST) (A(80));
31          IF SUBSTR(TEST,1,1)=B THEN DO;
33             PUT EDIT ('*****ERROR*****-THE FIRST PRODUCTION IN THE SYNTAX ',
               'DOES NOT HAVE A LEFTPART') (A,A) SKIP(2);
34             GO TO TERMINATE;
35          END;
36          GO TO FIRST;
37       CARD:
            GET EDIT (TEST) (A(80));
38       FIRST:
            TEST=TEST||B;
39          L1=2;
40          IF SUBSTR(TEST,1,1)=B THEN INPUT(1)=B;
42          ELSE DO;
43             L1=INDEX(TEST,B);
```

```
TESTPREC:

44                    IF L1 > 13 THEN PUT EDIT ('*****WARNING - THE SYMBOL ',
                      SUBSTR(TEST,1,L1-1),' HAS BEEN TRUNCATED TO 12 CHARACTERS')
                      (A,A,A) SKIP(2);
46                    INPUT(1)=SUBSTR(TEST,1,L1-1);
47                 END;
48                 L2=1;
49                 ELEMENT='';
50                 DO J=L1 TO 81 BY 1;
51                    A=SUBSTR(TEST,J,1);
52                    IF A¬=B THEN ELEMENT=ELEMENT||A;
54                    ELSE
54                    IF ELEMENT¬='' THEN DO;
56                       L2=L2+1;
57                       IF LENGTH(ELEMENT) > 12 THEN
58                       PUT EDIT ('*****WARNING - THE SYMBOL ',ELEMENT,
                         ' HAS BEEN TRUNCATED TO 12 CHARACTERS')
                         (A,A,A) SKIP(2);
59                       INPUT(L2)=ELEMENT;
60                       ELEMENT='';
61                    END;
62                 END;
63                 M=M+1;
64                 IF L2 > 7 THEN DO;
66                    PUT EDIT ('***** IN LINE ',M,' THERE ARE ',L2-1,
                      ' SYMBOLS ON THE RHS OF THE PRODUCTION. THE LIMIT IS 6.')
                      (A,F(3,0),A,F(2,0),A) SKIP(2);
67                    ERR1='1'B;
68                    L2=7;
69                 END;
70                 PUT EDIT (M,INPUT(1),'::=',(INPUT(J) DO J=2 TO L2))
                   (X(5),F(3,0),X(10),A(12),X(4),A(3),(6)(X(2),A(12))) SKIP;
71                 IF INPUT(1)=B THEN GO TO LAB2;
                   /*     PROCESS LEFT-SIDE OF PRODUCTION    */
73                 X=INPUT(1);
74                 DO I=1 TO N;   /* COMPARE WITH PREVIOUS SYMBOLS */
75                    IF X=SYMBOL(I) THEN GO TO LAB1;
77                 END;
78                 I,N=N+1; /* ADD NEW SYMBOL */
79                 IF N > 180 THEN ERR_MSG: DO;
81                    PUT EDIT ('***** AN IMPOSED LIMIT OF 180 UNIQUE SYMBOLS IN ',
                      'THE SYNTAX HAS BEEN EXCEEDED') (A,A) SKIP(2);
82                    GO TO TERMINATE;
83                 END;
84                 SYMBOL(I)=X;
85              LAB1:
86                 IF TERM(I) THEN NON=NON+1;
87                 TERM(I)='0'B;
88              LAB2:
                   /*     PROCESS RIGHT-SIDE OF PRODUCTION      */
                   DO J=2 TO L2;
89                    X=INPUT(J);
90                    DO K=1 TO N;
91                       IF X=SYMBOL(K) THEN GO TO LAB3;
93                    END;
94                    K,N=N+1;    /* ADD NEW SYMBOL */
95                    IF N > 180 THEN GO TO ERR_MSG;
97                    SYMBOL(N)=X;
```

TESTPREC:

```
 98                LA33:
 99                   IF J=2 THEN SUBSTR(LEFT,NUM*(I-1)+K,1)='1'B;
100                   ELSE
100                   DO;
101                      EQ=EQ+1;
102                      Y=NUM*(L-1)+K;
103                      SUBSTR(EQUAL,Y,1)='1'B;
104                      LINE(EQ,1)=Y;  /* STORES LINE #'S FOR ALL PAIRS */
105                      LINE(EQ,2)=M;  /* OF EQUAL PRECEDENCE          */
106                   END;
107                   L=K;
108                END;
109                SUBSTR(RIGHT,NUM*(I-1)+K,1)='1'B;
110                GO TJ CARD;

111             MATRIX:
                   BEGIN;
112                DCL X CHAR(1),PREC CHAR(N*N),(NONT(NON),K1,
                   TERMINAL(N-NON)) FIXED BIN(15);
                   /*            STORAGE AREAS
                        NONT      - STORES ALL NONTERMINAL SYMBOLS
                        TERMINAL - STORES ALL TERMINAL SYMBOLS
                        PREC      - PRECEDENCE MATRIX                          */
113                J,K=0;
114                PREC=' ';
115                DO I=1 TO N;
116                   IF TERM(I) THEN DO;
118                      J=J+1;
119                      TERMINAL(J)=I;
120                   END;
121                   ELSE
121                   DJ;
122                      K=K+1;
123                      NONT(K)=I;
124                   END;
125                END;
126                PUT EDIT ('NONTERMINAL SYMBOLS') (X(50),A) PAGE;
127                PUT EDIT ((SYMBOL(NONT(I)) DO I=1 TO K)) ((N)(A(12),X(10)))SKIP(2);
128                PUT EDIT ('TERMINAL SYMBOLS') (X(52),A) SKIP(4);
129                L=N-<;
130                PUT EDIT ((SYMBOL(TERMINAL(I)) DO I=1 TO J))
                   ((L)(A(12),X(10))) SKIP(2);
131                IF E??1 THEN DO;
133                   PUT EDIT ('***** SHORTEN THE PRODUCTIONS WHICH ARE TOO LONG ',
                      'AND RUN AGAIN') (A,A) SKIP(4);
134                   GJ TO TERMINATE;
135                END;
136                PUT EDIT ('VIOLATIONS') (X(55),A) SKIP(4);

                   /* THE STRINGS 'LEFT' AND 'RIGHT' ARE CHANGED TO INCLUDE ALL THE
                      LEFT-MOST DERIVABLE SYMBOLS AND RIGHT-MOST DERIVABLE SYMBOLS
                      RESPECTIVELY.  THIS IS ACCOMPLISHED THROUGH THE USE OF
                      WARSHALL'S ALGORITHM.
                      WARSHALL,S. A THEORFM ON BOOLEAN MATRICES.
                      J.ACM 9 (JAN.1962),11-12.                                */
137             WARSHALL:
                   DJ I=1 TO N;
```

TESTPREC:

```
                DO J=1 TO N;
                   IF SUBSTR(LEFT,NUM*(J-1)+I,1) THEN DO K=1 TO N;
                      IF SUBSTR(LEFT,NUM*(I-1)+K,1) THEN
                      SUBSTR(LEFT,NUM*(J-1)+K,1)='1'B;
                   END;
                   IF SUBSTR(RIGHT,NUM*(J-1)+I,1) THEN DO K=1 TO N;
                      IF SUBSTR(RIGHT,NUM*(I-1)+K,1) THEN
                      SUBSTR(RIGHT,NUM*(J-1)+K,1)='1'B;
                   END;
             END WARSHALL;

             /* DEVELOPMENT OF THE PRECEDENCE MATRIX USING THE PRECEDENCE
                DEFINITIONS DEFINED IN:
                WIRTH,N. AND WEBER,M. EULER: A GENERALIZATION OF ALGOL,
                AND ITS FORMAL DEFINITION; PART I. COMM. ACM 9 (JAN.1966)     */
          WIRTH:
             DO I=1 TO N;
                DO J=1 TO N;
                   IF SUBSTR(EQUAL,NUM*(I-1)+J,1) THEN DO;
                      /* DEVELOP THE RELATIONSHIP = */
                      X=SUBSTR(PREC,N*(I-1)+J,1);
                      IF X=B THEN
                      SUBSTR(PREC,N*(I-1)+J,1)='=';
                      ELSE DO;  /* ERROR MESSAGES */
                         PUT EDIT (SYMBOL(I),X,'=',SYMBOL(J))
                         (A(12),X(2),A,A,X(2),A(12)) SKIP(2);
                         IF X='<' THEN CALL LESS(I,J);
                         ELSE
                         CALL GREAT(I,J);
                         CALL LINE_NO(NUM*(I-1)+J,Z);
                         PUT EDIT ('NOTE: = BECAUSE',SYMBOL(I),'ADJACENT TO',
                         SYMBOL(J),'IN',Z)
                         (X(33),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,F(4,0))
                         SKIP;
                         ERR='1'B;
                      END;
                      IF TERM(J)='0'B THEN
                   S:  /* DEVELOP THE RELATIONSHIP < */
                      DO K=1 TO N;
                         IF SUBSTR(LEFT,NUM*(J-1)+K,1) THEN DO;
                            X=SUBSTR(PREC,N*(I-1)+K,1);
                            IF X=B THEN SUBSTR(PREC,N*(I-1)+K,1)='<';
                            ELSE
                            IF X¬='<' THEN DO;  /* ERROR MESSAGES */
                               PUT FILE(SYSPRINT) EDIT (SYMBOL(I),
                               X,'<',SYMBOL(K))
                               (A(12),X(2),A,A,X(2),A(12)) SKIP(2);
                               IF X='=' THEN DO;
                                  CALL LINE_NO(NUM*(I-1)+K,Z);
                                  PUT EDIT ('NOTE: = BECAUSE',SYMBOL(I),
                                  'ADJACENT TO',SYMBOL(K),'IN',Z)
                                  (X(3),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,
                                  F(4,0));
                               END;
                               ELSE
                               CALL GREAT(I,K);
                               CALL LINE_NO(NUM*(I-1)+J,Z);
```

```
TESTPREC:

184                  PUT EDIT ('NOTE: < BECAUSE',SYMBOL(K),
                     'IS LDS OF',SYMBOL(J),'&',SYMBOL(I),'=',
                     SYMBOL(J),'IN',Z)
                     (X(33),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,X(1)
                     ,A(12),X(1),A,X(1),A(12),X(1),A,F(4,0)) SKIP;
185                  ERR='1'B;
186               END;
187         END S;
189         IF TERM(I)='0'B THEN DO;
191         S1:  /* DEVELOP THE RELATIONSHIP > */
               DO K=1 TO N;
192               IF SUBSTR(RIGHT,NUM*(I-1)+K,1) THEN DO;
194                  X=SUBSTR(PREC,N*(K-1)+J,1);
195                  IF X=B THEN SUBSTR(PREC,N*(K-1)+J,1)='>';
197                  ELSE
197                  IF X¬= '>' THEN DO;  /* ERROR MESSAGES */
199                     PUT FILE(SYSPRINT) EDIT (SYMBOL(K),
                        X,'>',SYMBOL(J))
                        (A(12),X(2),A,A,X(2),A(12)) SKIP(2);
200                     IF X='=' THEN DO;
202                        CALL LINE_NO(NUM*(K-1)+J,Z);
203                        PUT EDIT ('NOTE: = BECAUSE',SYMBOL(K),
                           'ADJACENT TO',SYMBOL(J),'IN',Z)
                           (X(3),A,X(1),A(12),X(1),A,X(1),A(12),
                           X(1),A,F(4,0));
204                     END;
205                     ELSE
205                     CALL LESS(K,J);
206                     CALL LINE_NO(NUM*(I-1)+J,Z);
207                     PUT EDIT ('NOTE: > BECAUSE',SYMBOL(K),
                        'IS RDS OF',SYMBOL(I),'&',SYMBOL(I),
                        '=',SYMBOL(J),'IN',Z)
                        (X(33),A,X(1),A(12),X(1),A,
                        X(1),A(12),X(1),A,X(1),A(12),X(1),A,X(1),
                        A(12),X(1),A,F(4,0)) SKIP;
208                     ERR='1'B;
209                  END;
210            END S1;
212            IF TERM(J)='0'B THEN
213         S2:  /* DEVELOP THE RELATIONSHIP > */
               DO K=1 TO N;
214               IF SUBSTR(RIGHT,NUM*(I-1)+K,1) THEN DO L=1 TO N;
216                  IF SUBSTR(LEFT,NUM*(J-1)+L,1) THEN DO;
218                     X=SUBSTR(PREC,N*(K-1)+L,1);
219                     IF X=B THEN SUBSTR(PREC,N*(K-1)+L,1)='>';
221                     ELSE
221                     IF X ¬= '>' THEN DO; /* ERROR MESSAGES */
223                        PUT FILE(SYSPRINT) EDIT (SYMBOL(K),
                           X,'>',SYMBOL(L))
                           (A(12),X(2),A,A,X(2),A(12)) SKIP(2);
224                        IF X= '=' THEN DO;
226                           CALL LINE_NO(NUM*(K-1)+L,Z);
227                           PUT EDIT ('NOTE: = BECAUSE',SYMBOL(K),
                              'ADJACENT TO',SYMBOL(L),'IN',Z)
                              (X(3),A,X(1),A(12),X(1),A,X(1),A(12),
                              X(1),A,F(4,0));
228                        END;
```

```
      TESTPREC:

229                                        ELSE
229                                        CALL LESS(K,L);
230                                        CALL LINE_NO(NUM*(I-1)+J,Z);
231                                        PUT EDIT ('NOTE: > BECAUSE',SYMBOL(K),
                                           'IS RDS OF',SYMBOL(I),'&',SYMBOL(L),
                                           'IS LDS OF',SYMBOL(J)) (X(33),A,X(1),A(12),
                                           X(1),A,X(1),A(12),X(1),A,X(1),A(12),X(1),
                                           A,X(1),A(12))SKIP;
232                                        PUT EDIT ('&',SYMBOL(I),'=',SYMBOL(J),'IN',
                                           Z) (X(33),A,X(1),A(12),X(1),A,X(1),A(12),
                                           X(1),A,F(4,0)) SKIP;
233                                        ERR='1'B;
234                                     END;
235                              END S2;
238                           END;
239                END WIRTH;
242                IF ¬ERR THEN PUT EDIT ('NO PRECECENCE VIOLATIONS OCCURRED')
                   (X(44),A) SKIP(2);
244                IF PREC_MAT THEN GO TO CHECK;

                   /* PRINT THE PRECEDENCE MATRIX                                  */
246                PUT EDIT ('PRECEDENCE MATRIX') (X(52),A) PAGE;
247                IF N<=100 THEN DO;
249                   K=N/10;     K1=N;
251                END;
252                ELSE
252                DO;
253                   K=10;     K1=100;
255                END;
256                PUT EDIT (('|' DO J=1 TO K)) (X(18),(K)(X(9),A)) SKIP(2);
257                PUT EDIT (('.' DO J=1 TO K1)) (X(18),(K1)A(1)) SKIP(0);
258                J=1-N;
259                DO I=1 TO N;
260                   J=J+N;
261                   PUT EDIT (I,SYMBOL(I),'|',SUBSTR(PREC,J,K1))
                      (F(3,0),X(1),A(12),X(1),A(1),A) SKIP;
262                   IF N>100 THEN
263                   PUT EDIT ('|',SUBSTR(PREC,J+100,N-K1)) (X(17),A(1),A) SKIP;
264                END;
265                PUT EDIT (('|' DO J=1 TO K)) (X(18),(K)(X(9),A)) SKIP;
266                PUT EDIT (('.' DO J=1 TO K1)) (X(18),(K1)A(1)) SKIP(0);
267             CHECK:
268                IF ERR | FIND_F_G THEN GO TO TERMINATE;

                   /* CALCULATION OF F & G FUNCTIONS USING ALGORITHM 265 OF
                      COLLECTED ALGORITHMS FROM CACM BY NICLAUS WIRTH              */
269             FUNCTIONS:
                   BEGIN;
270                DCL (F(N),G(N),FMIN,GMIN) FIXED BIN(15),(LS,EQ,GR) CHAR(1);
271                DCL FIXROW ENTRY (FIXED BIN(15),FIXED BIN(15),FIXED BIN(15));
272                DCL FIXCOL ENTRY (FIXED BIN(15),FIXED BIN(15),FIXED BIN(15));
273                LS='<';
274                EQ='=';
275                GR='>';
276                K1=0;
277                F,G=0;
278                DO K=1 BY 1 TO N;
```

TESTPREC:

```
                    FMIN=1;
                    DO J=1 BY 1 TO K1;
                       X=SUBSTR(PREC,N*(K-1)+J,1);
                       IF X= GR & FMIN <= G(J) THEN FMIN=G(J)+1;
                       ELSE
                       IF X = EQ & FMIN < G(J) THEN FMIN=G(J);
                    END;
                    F(K)=FMIN;
                    DO J=K1 BY -1 TO 1;
                       X=SUBSTR(PREC,N*(K-1)+J,1);
                       IF X = LS & FMIN>= G(J) THEN CALL FIXCOL(K,J,1);
                       ELSE
                       IF X = EQ & FMIN > G(J) THEN CALL FIXCOL(K,J,0);
                    END;
                    K1=K1+1;
                    GMIN=1;
                    DO I=1 BY 1 TO K;
                       X=SUBSTR(PREC,N*(I-1)+K,1);
                       IF X = LS & F(I) >= GMIN THEN GMIN=F(I)+1;
                       ELSE
                       IF X = EQ & F(I) > GMIN THEN GMIN=F(I);
                    END;
                    G(K)=GMIN;
                    DO I=K BY -1 TO 1;
                       X=SUBSTR(PREC,N*(I-1)+K,1);
                       IF X = GR & F(I) <= GMIN THEN CALL FIXROW(I,K,1);
                       ELSE
                       IF X = EQ & F(I) < GMIN THEN CALL FIXROW(I,K,0);
                    END;
                 END;
                 PUT FILE(SYSPRINT) EDIT ('PRECEDENCE FUNCTIONS','F','G')
                 (A,X(6),A,X(5),A) PAGE;
                 DO I=1 TO N;
                    PUT FILE(SYSPRINT) EDIT (I,SYMBOL(I),F(I),G(I))
                    (F(3,0),X(4),A(12),X(5),F(3,0),X(3),F(3,0)) SKIP;
                 END;
              FIXROW: PROCEDURE (I,L,T) RECURSIVE;
                 DCL (J,I,L,T) FIXED BIN(15);
                 F(I)=G(L)+T;
                 IF K=K1 THEN DO;
                    X=SUBSTR(PREC,N*(I-1)+K,1);
                    IF X = LS & F(I) >= G(K) THEN GO TO NO_F_G;
                    ELSE
                    IF X = EQ & F(I) ¬= G(K) THEN GO TO NO_F_G;
                 END;
                 DO J=K1 BY -1 TO 1;
                    X=SUBSTR(PREC,N*(I-1)+J,1);
                    IF X = LS & F(I) >=G(J) THEN CALL FIXCOL(I,J,1);
                    ELSE
                    IF X = EQ & F(I) ¬= G(J) THEN CALL FIXCOL(I,J,0);
                 END;
                 END FIXROW;
              FIXCOL: PROCEDURE (L,J,T) RECURSIVE;
                 DCL (J,I,L,T) FIXED BIN(15);
                 G(J)=F(L)+T;
                 IF K ¬= K1 THEN DO;
                    X=SUBSTR(PREC,N*(K-1)+J,1);
```

```
            IF X = GR & F(K) <= G(J) THEN GO TO NO_F_G;
            ELSE
            IF X = EQ & F(K) ¬= G(J) THEN GO TO NO_F_G;
         END;
         DO I=K BY -1 TO 1;
            X=SUBSTR(PREC,N*(I-1)+J,1);
            IF X = GR & F(I) <= G(J) THEN CALL FIXROW(I,J,1);
            ELSE
            IF X = EQ & F(I) ¬= G(J) THEN CALL FIXROW (I,J,0);
         END;
         END FIXCOL;
         END FUNCTIONS;
         GO TO TERMINATE;
      NO_F_G:
         PUT EDIT ('NO F AND G FUNCTIONS EXIST') (X(48),A) SKIP;
         GO TO TERMINATE;
         END MATRIX;

         /* PROCEDURE 'LESS' IS USED TO GIVE ERROR MESSAGES FOR PRECEDENCE
            VIOLATIONS.
            IT DETERMINES THE ORIGIN OF THE RELATIONSHIP < IN 'PREC'      */
      LESS: PROCEDURE (X,Y);
         DCL (X,Y,I) FIXED BIN(15);
      L1:
         DO I=1 TO N;
            IF SUBSTR(LEFT,NUM*(I-1)+Y,1) THEN DO;
               IF SUBSTR(EQUAL,NUM*(X-1)+I,1) THEN DO;
                  CALL LINE_NO(NUM*(X-1)+I,Z);
                  PUT EDIT ('NOTE: < BECAUSE',SYMBOL(Y),'IS LDS OF',
                  SYMBOL(I),'&',SYMBOL(X),'=',SYMBOL(I),'IN',Z)
                  (X(3),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,X(1),A(12),
                  X(1),A,X(1),A(12),X(1),A,F(4,0));
                  GO TO L2;
         END L1;
      L2:
         RETURN;
         END LESS;

         /* THE PROCEDURE 'GREAT' IS USED TO GIVE ERROR MESSAGES FOR
            PRECEDENCE VIOLATIONS.
            IT DETERMINES THE ORIGIN OF THE RELATIONSHIP > IN 'PREC'      */
      GREAT: PROCEDURE (X,Y);
         DCL (X,Y,I,J) FIXED BIN(15);
      L3:
         DO I=1 TO N;
            IF SUBSTR(RIGHT,NUM*(I-1)+X,1) THEN DO;
               IF SUBSTR(EQUAL,NUM*(I-1)+Y,1) THEN DO;
                  CALL LINE_NO(NUM*(I-1)+Y,Z);
                  PUT EDIT ('NOTE: > BECAUSE',SYMBOL(X),'IS RDS OF',
                  SYMBOL(I),'&',SYMBOL(I),'=',SYMBOL(Y),'IN',Z)
                  (X(3),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,X(1),A(12),
                  X(1),A,X(1),A(12),X(1),A,F(4,0));
                  GO TO L5;
         END L3;
      L4:
         DO I=1 TO N;
            IF SUBSTR(RIGHT,NUM*(I-1)+X,1) THEN DO J=1 TO N;
```

```
            TESTPREC:

391                          IF SUBSTR(LEFT,NUM*(J-1)+Y,1) THEN DO;
393                             IF SUBSTR(EQUAL,NUM*(I-1)+J,1) THEN DO;
395                                CALL LINE_NO(NUM*(I-1)+J,Z);
396                                PUT EDIT ('NOTE: > BECAUSE',SYMBOL(X),'IS RDS OF',
                                   SYMBOL(I),'&',SYMBOL(Y),'IS LDS OF',SYMBOL(J))
                                   (X(3),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,X(1),
                                   A(12),X(1),A,X(1),A(12));
397                                PUT EDIT ('&',SYMBOL(I),'=',SYMBOL(J),'IN',Z)
                                   (X(33),A,X(1),A(12),X(1),A,X(1),A(12),X(1),A,F(4,0))
                                   SKIP;
398                                GO TO L5;
399                     END L4;
403                  L5:
                        RETURN;
404                     END GREAT;

                        /* PROCEDURE 'LINE_NO' GIVES THE LINE IN THE PRODUCTIONS IN
                           WHICH THE RELATION = WAS DEVELOPED                           */
405                  LINE_NO: PROCEDURE(X,Y);
406                     DCL (X,Y,I) FIXED BIN(15);
407                  L7:
                        DO I=1 TO EQ;
408                        IF LINE(I,1)=X THEN DO;
410                           Y=LINE(I,2);
411                           GO TO L8;
412                     END L7;
414                  L8:
                        RETURN;
415                     END LINE_NO;
416                  TERMINATE:
                        END TESTPREC;
```

APPENDIX B

LISTING OF EXTRACT

EXTRACT:

```
1            EXTRACT:
                PROCEDURE OPTIONS (MAIN);
2               DCL (INPUT(15),SYMBOL(180),A) CHAR(12) VAR,ERR1 BIT(1);
3               DCL (LEFT,RIGHT,EQUAL) BIT(32400),TERM(180) BIT(1),
                (X,BLANK) CHAR(1),(I,J,K,L,N,M,NUM,NON,LOC) FIXED BIN(15);
4               DCL TEST CHAR(81) VAR,ELEMENT CHAR(80) VAR, L2 FIXED BIN(15);
5               DCL TABLES FILE STREAM OUTPUT; /* INFORMATION FOR 'ANALYSE' */
6               OPEN FILE(SYSPRINT) PRINT LINESIZE(132) PAGESIZE(61);
7               OPEN FILE(TABLES) LINESIZE(100);
                /*           MAJOR DATA AREAS
                     SYMBOL - STORES THE SYMBOLS USED IN THE SYNTAX
                     EQUAL  - NOTES SYMBOLS WHICH OCCUR ADJACENT TO EACH OTHER ON
                              THE RIGHT-HAND SIDE OF PRODUCTIONS
                     LEFT   - NOTES LEFT-MOST SYMBOLS ON THE
                              RIGHT-HAND SIDE OF PRODUCTIONS
                     RIGHT  - NOTES RIGHT-MOST SYMBOLS ON THE
                              RIGHT-HAND SIDE OF PRODUCTIONS
                     TERM   - NOTES WHICH SYMBOLS ARE NONTERMINAL                */
8               NUM=180;
9               NON,LOC=0;
10              BLANK=' ';
11              LEFT,RIGHT,EQUAL='0'B;
12              TERM='1'B;
13              M=0;
14              N=0;
15              ERR1='0'B;

16            START1:
                BEGIN;
17              DCL (NUMB(500),PROD(300,3),LGTH_POINTER_1(6),LGTH_POINTER_2(6))
                FIXED BIN(15);
18              LGTH_POINTER_1,LGTH_POINTER_2=0;
                /*           DATA AREAS
                     NUMB   - STORES INTERNAL SYMBOL NUMBERS FOR ALL SYMBOLS
                              OCCURRING ON THE RHS OF PRODUCTIONS
                     PROD   - FOR I FROM 1 TO M:
                            PROD(I,1) - POINTS TO THE NEXT ROW IN 'PROD'  FOR
                                        A PRODUCTION OF THE SAME LENGTH
                            PROD(I,2) - POINTS TO THE LOCATION IN 'NUMB' WHERE THE
                                        RIGHT SIDE OF PRODUCTION I IS STORED
                            PROD(I,3) - INTERNAL SYMBOL NUMBER FOR LEFT SIDE
                                        OF PRODUCTION I.                         */
19              ON ENDFILE (SYSIN) GO TO WRITE;
21              PUT EDIT ('PRODUCTIONS') (X(54),A);
                /* READ IN PRODUCTIONS IN FREE FORMAT AND LOAD MAJOR DATA AREAS */
22              GET EDIT (TEST) (A(80));
23              IF SUBSTR(TEST,1,1)=BLANK THEN DO;
25                 PUT EDIT ('*****ERROR*****-THE FIRST PRODUCTION IN THE SYNTAX ',
                   'DOES NOT HAVE A LEFTPART') (A,A) SKIP(2);
26                 GO TO THE_END;
27              END;
28              GO TO FIRST;

29            CARD:
                GET EDIT (TEST) (A(80));
30            FIRST:
                TEST=TEST||BLANK;
```

```
EXTRACT:

              L1=2;
              IF SUBSTR(TEST,1,1)=BLANK THEN INPUT(1)=BLANK;
              ELSE DO;
                 L1=INDEX(TEST,BLANK);
                 IF L1 > 13 THEN PUT EDIT ('*****WARNING - THE SYMBOL ',
                 SUBSTR(TEST,1,L1-1),' HAS BEEN TRUNCATED TO 12 CHARACTERS')
                 (A,A,A) SKIP;
                 INPUT(1)=SUBSTR(TEST,1,L1-1);
              END;
              L2=1;
              ELEMENT='';
              DO J=L1 TO 81 BY 1;
                 X=SUBSTR(TEST,J,1);
                 IF X¬=BLANK THEN ELEMENT=ELEMENT||X;
                 ELSE
                 IF ELEMENT¬='' THEN DO;
                    L2=L2+1;
                    IF LENGTH(ELEMENT) > 12 THEN
                    PUT EDIT ('*****WARNING - THE SYMBOL ',ELEMENT,
                    ' HAS BEEN TRUNCATED TO 12 CHARACTERS')
                    (A,A,A) SKIP(2);
                    INPUT(L2)=ELEMENT;
                    ELEMENT='';
                 END;
              END;
              M=M+1;
              IF L2 > 7 THEN DO;
                 PUT EDIT ('***** IN LINE ',M,' THERE ARE ',L2-1,
                 ' SYMBOLS ON THE RHS OF THE PRODUCTION. THE LIMIT IS 6.')
                 (A,F(3,0),A,F(2,0),A) SKIP(2);
                 ERR1='1'B;
                 L2=7;
              END;
              PUT EDIT (M,INPUT(1),'::=',(INPUT(J) DO J=2 TO L2))
              (X(5),F(3,0),X(10),A(12),X(4),A(3),(6)(X(2),A(12))) SKIP;
              IF INPUT(1)=BLANK THEN GO TO LEFT_BLANK;
              /* PROCESS LEFT-SIDE OF PRODUCTION                          */
              A=INPUT(1);
              DO I=1 TO N;  /* COMPARE WITH PREVIOUS SYMBOLS */
                 IF A=SYMBOL(I) THEN GO TO OLD_SYMBOL_1;
              END;
              I,N=N+1;
              IF N > 180 THEN
           ERR_MSG:
              DO;
                 PUT EDIT ('***** AN IMPOSED LIMIT OF 180 UNIQUE SYMBOLS IN ',
                 'THE SYNTAX HAS BEEN EXCEEDED') (A,A) SKIP(2);
                 GO TO THE_END;
              END;
              SYMBOL(I)=A;
           OLD_SYMBOL_1:
              IF TERM(I) THEN NON=NON+1;
              TERM(I)='0'B;
           LEFT_BLANK:
              LGTH=L2-1;
              IF LGTH_POINTER_1(LGTH)=0 THEN
              LGTH_POINTER_1(LGTH),LGTH_POINTER_2(LGTH)=M;
```

```
EXTRACT:

 83              ELSE
 83              PROD(LGTH_POINTER_2(LGTH),1),LGTH_POINTER_2(LGTH)=M;
 84              PROD(M,2)=LOC+1;
 85              PROD(M,3)=I;
                 /* PROCESS RIGHT-SIDE OF PRODUCTION                              */
 86              DO J=2 TO L2; /* COMPARE WITH PREVIOUS SYMBOLS */
 87                 A=INPUT(J);
 88                 DO K=1 TO N;
 89                    IF A=SYMBOL(K) THEN GO TO OLD_SYMBOL_2;
 91                 END;
 92                 K,N=N+1;
 93                 IF N > 180 THEN GO TO ERR_MSG;
 95                 SYMBOL(N)=A;
 96              OLD_SYMBOL_2:
                    LOC=LOC+1;
 97                 NUMB(LOC)=K;
 98                 IF J=2 THEN SUBSTR(LEFT,NUM*(I-1)+K,1)='1'B;
100                 ELSE
100                 SUBSTR(EQUAL,NUM*(L-1)+K,1)='1'B;
101                 L=K;
102              END;
103              SUBSTR(RIGHT,NUM*(I-1)+K,1)='1'B;
104              GO TO CARD;

105           WRITE:
106              IF ERR1 THEN GO TO END_START1;
107              DO LGTH=1 TO 6;
108                 IF LGTH_POINTER_2(LGTH)¬=0 THEN
109                 PROD(LGTH_POINTER_2(LGTH),1)=0;
110              END;
                 /* WRITE INFORMATION IN 'TABLES'                                 */
111              PUT FILE(TABLES) EDIT (N,LOC,M)
                 (F(3,0),X(1),F(3,0),X(1),F(3,0));
112              PUT FILE(TABLES) EDIT  ((SYMBOL(I) DO I=1 TO N)) ((N)A(12));
113              PUT FILE(TABLES) EDIT ((NUMB(I) DO I=1 TO LOC)) ((LOC)F(3,0));
114              DO I=1 TO M;
115                 PUT FILE(TABLES)  EDIT (PROD(I,1),PROD(I,2),PROD(I,3))
                    (F(3,0),X(2),F(3,0),X(2),F(3,0));
116              END;
117              PUT FILE(TABLES) EDIT ((LGTH_POINTER_1(I) DO I=1 TO 6))
                 ((6)(F(3,0),X(2)));
118           END_START1:
                 END START1;
119              IF ERR1 THEN GO TO THE_END;
121           START2:
                 BEGIN;
122              DCL KEY_WORD(N-NON) CHAR(12),OPERATOR(N-NON) CHAR(1),
                 (ID,I1,I2,KEY_WORD_NO(N-NON),OPERATOR_NO(N-NON)) FIXED BIN(15);
123              DCL DOUBLE_OP(N-NON) CHAR(2),DOUBLE_OP_NO(N-NON) FIXED BIN(15);
124              DCL (ANYSTRING,VAR_TABLE,VAR_STRING,NUMERO) FIXED BIN(15);
125              DCL TEMP(180) FIXED BIN(15),POINTER(13) FIXED BIN(15);
126              I1,I2,ID=0;
127              ANYSTRING,VAR_TABLE,VAR_STRING,NUMERO=0;
128              TEMP=0;

129           LOAD:
                 DO I=1 TO N;
```

```
EXTRACT:

                  IF TERM(I) THEN DO;
                     A=SYMBOL(I);
                     IF A>= 'A' THEN TEMP(I)=LENGTH(A);
                     ELSE
                     DO;  /* DETERMINE ALL SINGL5 AND DOUBLE OPERATORS  */
                        IF LENGTH(A)=1 THEN DO;
                           I2=I2+1;
                           OPERATOR(I2)=A;
                           OPERATOR_NO(I2)=I;
                        END;
                        ELSE
                        DO;
                           ID=ID+1;
                           DOUBLE_OP(ID)=A;
                           DOUBLE_OP_NO(ID)=I;
                           IF LENGTH(A) > 2 THEN PUT EDIT
                           ('***** - THE SYMBOL ',A,' HAS BEEN TRUNCATED TO ',
                           '2 CHARACTERS') (A,A,A,A) SKIP(2);
                        END;
                     END;
               END LOAD;
               /* STORE ALL THE KEY WORDS IN ORDER FROM SHORTEST TO LONGEST    */
            IN_ORDER:
               DO J=1 TO 12;
                  POINTER(J)=I1+1;
                  DO I=1 TO N;
                     IF TEMP(I)=J THEN DO;
                        I1=I1+1;
                        KEY_WORD(I1)=SYMBOL(I);
                        KEY_WORD_NO(I1)=I;
                        /* DETERMINE THE RESERVED WORDS USED IN THE SYNTAX     */
                        IF SYMBOL(I)='VAR_TABLE' THEN VAR_TABLE=I;
                        IF SYMBOL(I)='NUMERO' THEN NUMERO=I;
                        IF SYMBOL(I)='ANYSTRING' THEN ANYSTRING=I;
                        IF SYMBOL(I)='VAR_STRING' THEN VAR_STRING=I;
               END IN_ORDER;
               POINTER(13)=I1+1;
               /* WRITE INFORMATION IN 'TABLES' */
               PUT FILE(TABLES) EDIT (ANYSTRING,VAR_STRING,VAR_TABLE,NUMERO,
               I1,I2,ID) ((7)(F(3,0)));
               DO I=1 TO I1;
                  PUT FILE(TABLES) EDIT (KEY_WORD(I),KEY_WORD_NO(I))
                  (A(12),X(2),F(3,0));
               END;
               PUT FILE(TABLES) EDIT ((POINTER(I) DO I=1 TO 13))
               ((13)(F(3,0),X(2)));
               DO I=1 TO I2;
                  PUT FILE(TABLES) EDIT (OPERATOR(I),OPERATOR_NO(I))
                  (A(1),X(2),F(3,0));
               END;
               DO I=1 TO ID;
                  PUT FILE(TABLES) EDIT (DOUBLE_OP(I),DOUBLE_OP_NO(I))
                  (A(2),X(2),F(3,0));
               END;
               END START2;

            MATRIX:
```

```
EXTRACT:

                BEGIN;
                DCL PREC CHAR(N*N); /* PRECEDENCE MATRIX */
                PREC=' ';

                /* THE STRINGS 'LEFT' AND 'RIGHT' ARE CHANGED TO INCLUDE ALL THE
                   LEFT-MOST DERIVABLE SYMBOLS AND RIGHT-MOST DERIVABLE SYMBOLS
                   RESPECTIVELY.  THIS IS ACCOMPLISHED THROUGH THE USE OF
                   WARSHALL'S ALGORITHM.
                   WARSHALL,S. A THEOREM ON BOOLEAN MATRICES.
                   J.ACM 9 (JAN.1962),11-12.                                    */
             WARSHALL:
                DO I=1 TO N;
                   DO J=1 TO N;
                      IF SUBSTR(LEFT,NUM*(J-1)+I,1) THEN DO K=1 TO N;
                         IF SUBSTR(LEFT,NUM*(I-1)+K,1) THEN
                         SUBSTR(LEFT,NUM*(J-1)+K,1)='1'B;
                      END;
                      IF SUBSTR(RIGHT,NUM*(J-1)+I,1) THEN DO K=1 TO N;
                         IF SUBSTR(RIGHT,NUM*(I-1)+K,1) THEN
                         SUBSTR(RIGHT,NUM*(J-1)+K,1)='1'B;
                      END;
                END WARSHALL;

                /* DEVELOPMENT OF THE PRECEDENCE MATRIX USING THE PRECEDENCE
                   DEFINITIONS DEFINED IN:
                   WIRTH,N. AND WEBER,M. EULER: A GENERALIZATION OF ALGOL,
                   AND ITS FORMAL DEFINITION; PART I. COMM. ACM 9 (JAN.1966)    */
             WIRTH:
                DO I=1 TO N;
                   DO J=1 TO N;
                      IF SUBSTR(EQUAL,NUM*(I-1)+J,1) THEN DO;
                         IF SUBSTR(PREC,N*(I-1)+J,1)=' ' THEN
                         SUBSTR(PREC,N*(I-1)+J,1)='=';
                         ELSE
                         GO TO ERROR;
                         IF TERM(J)='0'B THEN
                      S:
                         DO K=1 TO N;
                            IF SUBSTR(LEFT,NUM*(J-1)+K,1) THEN DO;
                               X=SUBSTR(PREC,N*(I-1)+K,1);
                               IF X=BLANK THEN SUBSTR(PREC,N*(I-1)+K,1)='<';
                               ELSE
                               IF X¬='<' THEN GO TO ERROR;
                         END S;
                         IF TERM(I)='0'B THEN DO;
                         S1:
                            DO K=1 TO N;
                               IF SUBSTR(RIGHT,NUM*(I-1)+K,1) THEN DO;
                                  X=SUBSTR(PREC,N*(K-1)+J,1);
                                  IF X=BLANK THEN SUBSTR(PREC,N*(K-1)+J,1)='>';
                                  ELSE
                                  IF X¬= '>' THEN GO TO ERROR;
                            END S1;
                            IF TERM(J)='0'B THEN S2: DO K=1 TO N;
                               IF SUBSTR(RIGHT,NUM*(I-1)+K,1) THEN DO L=1 TO N;
                                  IF SUBSTR(LEFT,NUM*(J-1)+L,1) THEN DO;
                                     X=SUBSTR(PREC,N*(K-1)+L,1);
```

```
EXTRACT:

                                        IF X=BLANK THEN SUBSTR(PREC,N*(K-1)+L,1)='>';
                                        ELSE
                                        IF X ¬= '>' THEN GO TO ERROR;
                                 END S2;
                              END;
                  END WIRTH;
                  /* WRITE THE PRECEDENCE MATRIX IN 'TABLES' */
                  J=1-N;
                  DO I=1 TO N;
                     J=J+N;
                     PUT FILE(TABLES) EDIT (SUBSTR(PREC,J,N)) (A(N));
                  END;
                  PUT EDIT ('TABLES LOADED SUCCESSFULLY') (A) SKIP(4);
                  GO TO FINISH;
               ERROR:
                  PUT EDIT ('A PRECEDENCE VIOLATION OCCURRED') (A) SKIP(4);
                  ERR1='1'B;
               FINISH:
                  END MATRIX;
               THE_END:
                  IF ERR1 THEN
                  PUT EDIT ('USE THE PGM T E S T P R E C TO TEST YOUR LANGUAGE ',
                  'BEFORE ATTEMPTING TO LOAD THE TABLES') (A,A) SKIP(4);
                  CLOSE FILE(TABLES);
                  END EXTRACT;
```

APPENDIX C

LISTING OF ANALYSE

ANALYSE:

```
1      ANALYSE:
          PROCEDURE (PARM) OPTIONS(MAIN);
2         DCL PARM CHAR(100) VAR;
3         DCL (N,L,LOC,M,I,J,K,PT,PT1,LGTH,LINE) FIXED BIN(15);
4         DCL COUNT CHAR(1) VAR;
5         DCL (ERR1,ERR2,ERR3,ERR4,ERR5,ERR6,ERR7,TRACE) BIT(1),
          (DEL,SCAN_ERR) BIT(1) EXTERNAL;
6         DCL (FIX_UP(50,3) FIXED BIN(15),FIX_BIT(50,3) BIT(1)) EXTERNAL;
7         DCL (DEL_PRS_CT(20),DEL_PRS(20,2),NO_POINTER(20),
          DEL_>T,NO_PT,NO) FIXED BIN(15);
8         DCL TABLES FILE STREAM INPUT;
9         NO,NJ_PT,DEL_PT=0;
10        DEL_PRS_CT=0;
11        DEL,ERR1,ERR2,ERR3,ERR4,ERR5,ERR6,ERR7,TRACE,SCAN_ERR=0;
12        OPEN FILE(TABLES); /* INFORMATION ABOUT LANGUAGE FROM 'EXTRACT' */
13        LINE=1;
          /* CHECK 'PARM' FOR PARAMETERS BEING PASSED TO 'ANALYSE'        */
14        IF LENGTH(PARM) = 0 THEN GO TO START1;
16        I=INDEX(PARM,'OPT=');
17        IF I > 0 THEN DO;
19           IF SUBSTR(PARM,I+4,1)='1' THEN ERR1='1'B;
21           ELSE
21           E>R2='1'B;
22        END;
23        IF INDEX(PARM,'COUNT(') > 0 THEN
24        COUNT=SUBSTR(PARM,INDEX(PARM,'COUNT(')+6,1);
25        IF INDEX(PARM,'TRACE')>0 THEN TRACE='1'B;
27        GET FILE(TABLES) EDIT (N,LOC,M)
          (F(3,0),X(1),F(3,0),X(1),F(3,0));
          /* F>R A DESCRIPTION OF ALL THE INFORMATION IN 'TABLES'
             SEE CHAPTER 3, SECTION 2                                     */
28     START1:
          BEGIN;
29        DCL SYMBOL(N) CHAR(12),(NUMB(LOC),PROD(M,3),VAR_STRING,VAR_TABLE,
          NUMERO,ANYSTRING,SIZE,I1,I2,ID,LGTH_POINTER(6)) FIXED BIN(15);
30        DO I=1 TO N;
31           GET FILE(TABLES) EDIT (SYMBOL(I)) (A(12));
32        END;
33        DO I=1 TO LOC;
34           GET FILE(TABLES) EDIT (NUMB(I)) (F(3,0));
35        END;
36        DO I=1 TO M;
37           GET FILE(TABLES) EDIT (PROD(I,1),PROD(I,2),PROD(I,3))
             (F(3,0),X(2),F(3,0),X(2),F(3,0));
38        END;
39        GET FILE(TABLES) EDIT ((LGTH_POINTER(I) DO I=1 TO 6))
          ((6)(F(3,0),X(2)));
40        GET FILE(TABLES) EDIT (ANYSTRING,VAR_STRING,VAR_TABLE,NUMERO,
          I1,I2,ID) ((7)(F(3,0)));

41        IF VAR_TABLE > 0 THEN SIZE=250;
43        ELSE
43        SIZE=1;

44     START2:
          BEGIN;
45        DCL KEY_WORD(I1) CHAR(12),OPERATOR(I2) CHAR(1),
```

ANALYSE:

```
        (KEY_WORD_NO(I1),OPERATOR_NO(I2),POINTER(13)) FIXED BIN(15);
46      DCL DOUBLE_OP(ID) CHAR(2),DOUBLE_OP_NO(ID) FIXED BIN(15);
47      DCL PREC CHAR(N*N);
48      DCL (VARIABLE_1(SIZE),VAR_PT_1(12),VAR_PT_2(12),PGRAM,
        PT3,STACK(50),SAVE(30),I3,I4,TOT1) FIXED BIN(15),
        VARIABLE(SIZE) CHAR(12) VAR;
49      DCL IN CHAR(81) VAR, STRING CHAR(256) VAR,ERROR CHAR(80),
        X CHAR(12) VAR,(Y,W,QUOTE,BLANK) CHAR(1),(COMMENT,STR) BIT(1);
50      DCL (INFO,STATUS) FIXED BIN(15),Z CHAR(2);

51      DO I=1 TO I1;
52         GET FILE(TABLES) EDIT (KEY_WORD(I),KEY_WORD_NO(I))
           (A(12),X(2),F(3,0));
53      END;
54      GET FILE(TABLES) EDIT ((POINTER(I) DO I=1 TO 13))
        ((13)(F(3,0),X(2)));
55      DO I=1 TO I2;
56         GET FILE(TABLES) EDIT (OPERATOR(I),OPERATOR_NO(I))
           (A(1),X(2),F(3,0));
57      END;
58      DO I=1 TO ID;
59         GET FILE(TABLES) EDIT (DOUBLE_OP(I),DOUBLE_OP_NO(I))
           (A(2),X(2),F(3,0));
60      END;

61      J=1-N;
62      DO I=1 TO N;
63         J=J+N;
64         GET FILE(TABLES) EDIT (SUBSTR(PREC,J,N)) (A(N));
65      END;
66      CLOSE FILE(TABLES);
67      OPEN FILE(SYSPRINT) PRINT LINESIZE(132);
68      ON ENDFILE(SYSIN) GO TO EOF;
70      PUT EDIT ('ISN','SOURCE LISTING') (X(2),A,X(48),A) SKIP;
71      PUT SKIP(2);
72      STATUS,INFO=0;
73      VARIABLE_1=0;
74      VAR_PT_1=0;
75      STR,COMMENT='0'B;
76      QUOTE='''';
77      TOT1=0;
78      BLANK=' ';
79      ERROR=BLANK;
80      PT3,PT1=1;
81      SAVE(1)=0;
        /* EVERYTHING UP TO THIS POINT IS INITIALIZATION FOR THE RUN.
           FROM HERE ON THE SOURCE PROGRAM IS BEING PROCESSED          */
82    CYCLE:
        GET EDIT (IN)  (A(80)); /* CARD OF SOURCE PROGRAM */
83      PUT EDIT (IN) (X(25),A(80)) SKIP;
84      IF COUNT¬= '' THEN PUT EDIT (LINE) (X(2),F(3,0)) SKIP(0);
86      IN=IN||BLANK;
87      PT=0;
88      IF COMMENT THEN GO TO FLUSH_COMMENT;
90      IF STR THEN GO TO CHAR_STRINGS;
92    SCAN:
        PT=PT+1;
```

```
ANALYSE:

93         IF PT>80 THEN GO TO CYCLE; /* READ ANOTHER CARD */
95         Y=SUBSTR(IN,PT,1);
96         IF Y=BLANK THEN GO TO SCAN; /* REPEAT UNTIL NON-BLANK SYMBOL */
98         IF Y>= 'A' THEN GO TO LETTERS; /* EITHER LETTER OR DIGIT */
100        IF Y=COUNT THEN LINE=LINE+1; /* INCREMENTING ISN */
102        W=SUBSTR(IN,PT+1,1);
103        IF W < 'A' & W ¬=BLANK THEN DO; /* CHECK ALL DOUBLE OPERATORS */
105           Z=SUBSTR(IN,PT,2);
106           DO I=1 TO ID;
107              IF DOUBLE_OP(I) = Z THEN DO;
109                 PGRAM=DOUBLE_OP_NO(I);
110                 I3=PT;
111                 PT=PT+1;
112                 GO TO PARSE;
113              END;
114           END;
115        END;
           /* CHECK ALL SINGLE OPERATORS */
116        DO I=1 TO I2;
117           IF OPERATOR(I)=Y THEN DO;
119              PGRAM=OPERATOR_NO(I);
120              I3=PT;
121              GO TO PARSE;
122           END;
123        END;
124        IF Y¬=QUOTE THEN GO TO ERR_MSG;
126        STRING='';
           /* PROCESS STRINGS WITHIN QUOTES. - 'ANYSTRING' USED IN SYNTAX  */
127     CHAR_STRINGS:
128        IF ANYSTRING=0 THEN GO TO ERR_MSG;
129        I3=PT;
130        PT=PT+1;
131        I=INDEX(SUBSTR(IN,PT),QUOTE);
132        IF I=0 THEN DO;
134           STRING=STRING||SUBSTR(IN,PT,81-PT);
135           STR='1'B;
136           GO TO CYCLE;
137        END;
138        STRING=STRING||SUBSTR(IN,PT,I-1);
139        PT=I+PT-1;
140        STR='0'B;
141        PGRAM=ANYSTRING;
142        GO TO PARSE;
           /* INVALID OPERATOR */
143     ERR_MSG:
           PUT EDIT ('***** THE OPERATOR',Y,
           'WAS USED BUT DOES NOT APPEAR IN THE PRODUCTIONS. - DELETED')
           (A,X(1),A(1),X(1),A) SKIP;
144        SCAN_ERR='1'B;
145        GO TO SCAN;
146     LETTERS:
147        IF Y>= '0' THEN GO TO DIGITS;
148        I3=PT;
           /* KEY_WORD OR IDENTIFIER */
149     NEXT_LETTER:
           PT=PT+1;
150        IF SUBSTR(IN,PT,1)>= 'A' THEN GO TO NEXT_LETTER;
```

```
ANALYSE:

152             X=SUBSTR(IN,I3,PT-I3);
153             LGTH=LENGTH(X);
154             IF LGTH > 12 THEN GO TO VAR;
                /* CHECK KEY_WORDS */
156             IF POINTER(LGTH)=POINTER(LGTH+1) THEN GO TO NONE_THAT_LENGTH;
158             DO I= POINTER(LGTH) TO POINTER(LGTH+1)-1;
159                IF KEY_WORD(I)=X THEN DO;
161                   PGRAM=KEY_WORD_NO(I);
162                   PT=PT-1;
163                   GO TO PARSE;
164                END;
165             END;
166          NONE_THAT_LENGTH:
                /* PROCESS COMMENTS */
167             IF X¬= 'COMMENT' THEN GO TO VAR;
168             COMMENT='1'B;
169             PT=PT-1;
170          FLUSH_COMMENT:
                PT=PT+1;
171             IF INDEX(SUBSTR(IN,PT),';')=0 THEN GO TO CYCLE;
173             PT=INDEX(SUBSTR(IN,PT),';')+PT-1;
174             COMMENT='0'B;
175             GO TO SCAN;
                /* PROCESS IDENTIFIERS */
176          VAR:
                STRING=SUBSTR(IN,I3,PT-I3);
177             PT=PT-1;
178             IF VAR_STRING > 0 THEN DO; /* 'VAR_STRING USED IN SYNTAX  */
180                PGRAM=VAR_STRING;
181                GO TO PARSE;
182             END;
183             IF VAR_TABLE=0 THEN DO;
185                PUT EDIT ('***** - A VARIABLE ',STRING,' WAS FOUND ',
                   'BUT NEITHER VAR_STRING NOR VAR_TABLE APPEARS IN THE SYNTAX')
                   (A,A,A,A) SKIP(4);
186                GO TO THE_END;
187             END;
                /* 'VAR_TABLE' USED IN THE SYNTAX */
188             PGRAM=VAR_TABLE;
189             I=VAR_PT_1(LGTH);
190             GO TO TEST_VAR;
                /* CHECK WITH IDENTIFIERS ALREADY FOUND */
191          NEXT_VAR:
                I=VARIABLE_1(I);
192          TEST_VAR:
193             IF I=0 THEN GO TO VAR_NOT_FOUND;
194             IF VARIABLE(I)=X THEN GO TO VAR_FOUND;
196             GO TO NEXT_VAR;
                /* ADD NEW IDENTIFIER TO 'VARIABLE' */
197          VAR_NOT_FOUND:
                I,TOT1=TOT1+1;
198             IF TOT1 > SIZE  THEN DO;
200                PUT EDIT ('***** - THE ANALYSER LIMIT ON THE NUMBER OF ',
                   'VARIABLES ALLOWED, HAS BEEN EXCEEDED') (A,A) SKIP(4);
201                PUT EDIT ('EITHER REDUCE THE # OF VARIABLE NAMES USED OR ',
                   'REPLACE VAR_TABLE BY VAR_STRING IN THE SYNTAX AND HANDLE ',
                   'THE VARIABLES IN INTERPRET') (A,A,A) SKIP(2);
```

ANALYSE:

```
            GO TO THE_END;
         END;
         VARIABLE(TOT1)=X;
         IF VAR_PT_1(LGTH)=0 THEN VAR_PT_1(LGTH)=TOT1;
         ELSE
         VARIABLE_1(VAR_PT_2(LGTH))=TOT1;
         VAR_PT_2(LGTH)=TOT1;
      VAR_FOUND:.
         INFO=I;
         GO TO PARSE;
      DIGITS:
         I3=PT;
         IF NUMERO=0 THEN DO; /* CHECK DIGITS AGAINST KEY_WORDS */
            X=SUBSTR(IN,PT,1);
            LGTH=1;
            IF POINTER(LGTH)=POINTER(LGTH+1) THEN GO TO ERR_MSG_1;
            DO I=POINTER(LGTH) TO POINTER(LGTH+1)-1;
               IF KEY_WORD(I)=X THEN DO;
                  PGRAM=KEY_WORD_NO(I);
                  GO TO PARSE;
               END;
            END;
         ERR_MSG_1:
            PUT EDIT ('***** THE SYMBOL ',X,'WAS FOUND BUT NEITHER THIS ',
            'SYMBOL NOR NUMERO OCCURS IN THE SYNTAX') (A,A(1),A,A) SKIP;
            SCAN_ERR='1'B;
            GO TO SCAN;
         END;
         /* 'NUMERO' USED - PICKS OUT ENTIRE INTEGER */
      NEXT_DIGIT:
         PT=PT+1;
         IF SUBSTR(IN,PT,1)>= '0' THEN GO TO NEXT_DIGIT;
         STRING=SUBSTR(IN,I3,PT-I3);
         PGRAM=NUMERO;
         PT=PT-1;
         GO TO PARSE;

      EOF:
         IF ERR3|ERR1|ERR6 THEN DO;
            PUT EDIT ('***** - END OF FILE') (A) SKIP(2);
            GO TO THE_END;
         END;
         ERR3='1'B;
         GO TO JUMP;
      PARSE:
         IF ERR4 THEN GO TO RELATION;
         /* LOAD FIRST SYMBOL INTO 'STACK' */
         ERR4='1'B;
         STACK(1)=PGRAM;
         GO TO SCAN;
         /* FIND RELATIONSHIP BETWEEN TOP OF 'STACK' AND INCOMING SYMBOL */
      RELATION:
         I=N*(STACK(PT1)-1)+PGRAM;
         Y=SUBSTR(PREC,I,1);
         /* IF REQUESTED THEN PRINT TRACE OF PARSE */
         IF TRACE THEN PUT EDIT (SYMBOL(STACK(PT1)),SYMBOL(PGRAM),Y)
         (X(105),A(12),X(1),A(12),X(1),A(1)) SKIP;
```

```
         ANALYSE:

252               IF Y=BLANK THEN DO;  /* NO RELATIONSHIP - SYNTAX ERROR */
254                  SUBSTR(ERROR,I3,1)='$';
255                  PUT EDIT ('*****SYNTAX*****',ERROR) (A,X(9),A(80))SKIP;
256                  SUBSTR(ERROR,I3,1)=BLANK;
257                  SCAN_ERR='1'B;
258                  IF ERR6 THEN GO TO SET;
260                  GO TO SCAN;
261               END;
262               IF ERR1 THEN DO;  /* ONLY SCAN CONTINUING */
264                  STACK(PT1)=PGRAM;
265                  GO TO SCAN;
266               END;
267               IF ERR6 THEN GO TO DUMP_SCAN; /* ERROR RECOVERY IN PROGRESS */
269               IF Y='>' THEN
270            JUMP:
                  DO; /* LEFT-MOST REDUCIBLE SUBSTRING HAS BEEN ISOLATED */
271                  LGTH=PT1-SAVE(PT3);
272                  IF LGTH > 6 THEN DO;  /* LONGER THAN ANY PRODUCTION */
274                     I=0;
275                     GO TO FAIL;
276                  END;
                     /* DETERMINE WHICH PRODUCTION HAS BEEN FOUND */
277                  I=LGTH_POINTER(LGTH);
278                  GO TO TEST;
279               NEXT:
                     I=PROD(I,1);
280               TEST:
281                  IF I=0 THEN GO TO FAIL;
282                  DO J=1 TO LGTH;
283                     IF NUMB(PROD(I,2)+J-1)¬= STACK(SAVE(PT3)+J) THEN GO TO NEXT;
285                  END;
                     /* DOES NOT MATCH ANY PRODUCTION OF THE LANGUAGE */
286               FAIL:
                     J=SAVE(PT3)+1;
287                  IF I=0 THEN PUT EDIT ('***** - INVALID STACK SEQUENCE-',
                     (SYMBOL(STACK(K)) DO K=J TO PT1))
                     (A,X(5),(PT1-J+1)(A(12),X(1))) SKIP(2);
289                  IF ERR2 THEN GO TO BY_PASS; /* DON'T CALL 'INTERPRET' */
291                  CALL INTERPRET(I,J,PT1,STACK,VARIABLE,SYMBOL,STRING,INFO,
                     STATUS);
292                  IF STATUS > 0 THEN DO; /* 'INTERPRET' HAS ALTERED 'STATUS' */
294                     IF STATUS=3 THEN GO TO THE_END; /* TERMINATE */
296                     IF STATUS=1 THEN DO; /* CONTINUE SCAN ONLY */
298                        ERR1='1'B;
299                        PUT EDIT ('***** - ONLY SCAN CONTINUING') (A) SKIP(2);
300                        GO TO SCAN;
301                     END;
302                     ELSE
302                     DO;
                           /* SCAN AND PARSE BUT DON'T CALL 'INTERPRET' */
303                        ERR2='1'B;
304                        PUT EDIT ('***** - PARSE CONTINUING BUT INTERPRET NO ',
                           'LONGER CALLED') (A,A) SKIP(2);
305                     END;
306                  END;
307               BY_PASS:
                     PT1=J;
```

```
ANALYSE:

308                IF I > 0 THEN DO;  /*  MAKE THE REDUCTION */
310                   STACK(J)=PROD(I,3);
311                   I=N*(STACK(J-1)-1)+STACK(J);
312                   IF SUBSTR(PREC,I,1) = '<' THEN GO TO RELATION;
314                   PT3=PT3-1;
315                   IF PT3 > 0 THEN GO TO RELATION;
317                   PUT EDIT ('***** - PARSE TERMINATED BY SYNTAX ANALYSER. ',
                      'SCAN CONTINUING') (A,A) SKIP(2);
318                   ERR1='1'B;
319                   STACK(PT1)=PGRAM;
320                   GO TO SCAN;
321                END;
322                IF ¬DEL THEN GO TO NO_REC; /* NO ERROR RECOVERY */
324                IF ¬ERR5 THEN GO TO LOAD_TAB; /* FIRST ATTEMPT AT ERROR REC. */
326             ERR_RECOVERY:
                   ERR6='1'B;
327                PUT EDIT ('SCAN DELETED FROM HERE-->') (X(I3-1),A) SKIP;
328                K=PT1;
329             DECREASE:  /* SEARCH FOR SYMBOLS IN THE 'STACK' */
                   K=K-1;
330                IF K<=0 THEN GO TO ERR_MSG_2;
332                DO I4=1 TO NO_PT;
333                   IF STACK(K)=FIX_UP(NO_POINTER(I4),1) THEN GO TO DUMP_SCAN;
335                END;
336                GO TO DECREASE;
337             SET:
                   ERR7='1'B;
338             DUMP_SCAN:
                   /* KEEP TRACK OF DELETE-DELETE PAIRS */
                   DO M=1 TO DEL_PT;
339                   IF DEL_PRS(M,1)=PGRAM THEN DEL_PRS_CT(M)=DEL_PRS_CT(M)+1;
341                   IF DEL_PRS(M,2)=PGRAM THEN DEL_PRS_CT(M)=DEL_PRS_CT(M)-1;
343                END;
                   /* CHECK SYMBOLS FROM SCAN */
344                DO L=NO_POINTER(I4) TO NO_POINTER(I4+1)-1;
345                   IF PGRAM=FIX_UP(L,2) THEN GO TO FOUND;
347                END;
348             L1:
349                IF ERR7 THEN ERR7='0'B;
350                ELSE
350                STACK(PT1)=PGRAM;
351                GO TO SCAN;
352             FOUND:  /* CHECK THAT DELETE-DELETE PAIRS ARE MATCHED */
                   DO M=1 TO DEL_PT;
353                   IF DEL_PRS_CT(M) > 0 THEN GO TO L1;
355                END;
356                IF FIX_BIT(L,2) THEN I3=PT+1;
358                PUT EDIT ('<--TO HERE') (X(24+I3),A) SKIP;
359                IF FIX_UP(L,3)¬=0 THEN DO; /* DELETE DEEPER IN 'STACK' */
361                L2:
                      K=K-1;
362                   IF K<=0 THEN GO TO ERR_MSG_2;
364                   IF STACK(K)¬=FIX_UP(L,3) THEN GO TO L2;
366               END;
367               IF FIX_BIT(L,3)|FIX_BIT(L,1) THEN K=K-1;
369               IF K<=0 THEN GO TO ERR_MSG_2;
371                PUT EDIT ('***** - DELETED FROM STACK-',
```

```
                 (SYMBOL(STACK(M)) DO M=K+1 TO PT1-1))
                 (A,X(5),(PT1+K+1)(A(12),X(1))) SKIP;
372              ERR6='0'B;
373              PT1=K;
                 /* ADJUST 'SAVE' AFTER DELETIONS FROM 'STACK' */
374              DO I=1 TO PT3;
375                 IF SAVE(I)>=PT1 THEN DO;
377                 .  PT3=I-1;
378                    IF PT3<=0 THEN GO TO ERR_MSG_2;
380                    GO TO L3;
381                 END;
382              END;
383           L3:
                 DEL_PRS_CT=0;
384              IF FIX_BIT(L,2) THEN GO TO SCAN;
386              GO TO RELATION;
                 /* LOAD THE ARRAYS 'NO_POINTER' AND 'DEL_PRS' WHEN ERROR
                    RECOVERY IS ATTEMPTED FOR THE FIRST TIME                 */
387           LOAD_TAB:
                 NO=NO+1;
388              IF FIX_UP(NO,2)=0 THEN DO;
390                 NO_POINTER(NO_PT+1)=NO;
391                 ERR5='1'B;
392                 GO TO ERR_RECOVERY;
393              END;
394              IF FIX_UP(NO,1)¬= 0 THEN DO;
396                 NO_PT=NO_PT+1;
397                 NO_POINTER(NO_PT)=NO;
398              END;
399              IF FIX_BIT(NO,2) THEN DO;
401                 IF FIX_BIT(NO,1)|FIX_BIT(NO,3) THEN DO;
403                    DEL_PT=DEL_PT+1;
404                    DEL_PRS(DEL_PT,2)=FIX_UP(NO,2);
405                    IF FIX_BIT(NO,1) THEN
406                    DEL_PRS(DEL_PT,1)=FIX_UP(NO_POINTER(NO_PT),1);
407                    ELSE
407                    DEL_PRS(DEL_PT,1)=FIX_UP(NO,3);
408                 END;
409              END;
410              GO TO LOAD_TAB;
411           ERR_MSG_2:  /* COULD NOT FIND REQUIRED SYMBOL IN 'STACK' */
                 PUT EDIT ('***** - ERROR RECOVERY FAILED - ONLY ',
                 'SCAN CONTINUING') (A,A) SKIP(2);
412              ERR1='1'B;
413              ERR6='0'B;
414              STACK(PT1)=PGRAM;
415              GO TO SCAN;
416           NO_REC:  /* USER DID NOT PROVIDE ERROR RECOVERY */
                 ERR1='1'B;
417              PUT EDIT ('***** - NO ERROR RECOVERY PROVIDED - ONLY ',
                 'SCAN CONTINUING') (A,A) SKIP(2);
418              STACK(PT1)=PGRAM;
419              GO TO SCAN;
420           END;
421           IF Y='<' THEN DO;  /* THE RELATIONSHIP IS < */
423              PT3=PT3+1;
424              SAVE(PT3)=PT1;
```

ANALYSE:

```
               END;
               PT1=PT1+1;
               STACK(PT1)=PGRAM;
               GO TO SCAN;
            THE_END:
               END ANALYSE;
```

APPENDIX D

REFERENCES

1. Wirth, N., and Weber, H. EULER A Generalization of ALGOL, and its Formal Definition: Pt. I. ACM 9 (Jan. 1966), 13-23, 25

2. Floyd, R. W. Syntactic Analysis and Operator Precedence. J. ACM 10,3 (July 1963), 316-333

3. Warshall, S. A Theorem on Boolean Matrices. J. ACM, 9,1 (Jan. 1962), 11-12

4. Wirth, N. Algorithm 265: Finding Precedence Functions. Comm. ACM 8,10 (Oct. 1965) 604-605

5. Martin, D. F. Boolean Matrix Methods for the Detection of Simple Precedence Grammars. Comm. ACM 11,10 (Oct. 1968) 685-687

6. Bell, J. R. A New Method for Determining Linear Precedence Functions for Precedence Grammars. Comm. ACM 12,10 (Oct. 1969) 567-569