SLAC-133
STAN-CS-71-226
UC-32
(MISC)

SIMPLE - - A SIMPLE PRECEDENCE TRANSLATOR WRITING SYSTEM *

JAMES E. GEORGE
STANFORD LINEAR ACCELERATOR CENTER
AND
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
Stanford, California

PREPARED FOR THE U.S. ATOMIC ENERGY
COMMISSION UNDER CONTRACT No. AT(04-3)-515

July 1971

Reproduced in the USA. Available from the Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia 22151.
Price: Full size copy $3.00; microfiche copy $0.95.

*Supported in part by the National Science Foundation, Contract No. 2SFGJ687.

ABSTRACT

SIMPLE is a translator writing system composed of a simple precedence syntax analyzer and a semantic constructor and is implemented in PL/I. It provides an error diagnostic and recovery mechanism for any system implemented using SIMPLE. The removal of precedence conflicts is discussed in detail with several examples.

The utilization of SIMPLE is illustrated by defining a command language meta system for the construction of scanners for a wide variety of command oriented languages. This meta system is illustrated by defining commands from several text editors.

TABLE OF CONTENTS

		Page
1.	Introduction	1
2.	Input Data to Simple's Executive	5
3.	Syntax Analyzer and Parser	8
	3.1 Definitions and Notation	9
	3.2 Transforming a Grammar to Simple Precedence	10
	3.2.1 Removing Precedence Conflicts	11
	3.2.2 Transforming a S-Precedence Grammar to Simple Precedence	18
	3.2.3 Transformation Examples	18
	3.3 Input Conventions for the Syntax Analyzer	27
	3.4 Syntax Analyzer Output	28
	3.5 Parser	29
	3.5.1 Declarations in the Parser	32
	3.5.2 Declarations and Initialization Inserted by the Syntax Analyzer	32
	3.5.3 Symbol Recognition	34
	3.5.4 Parsing	35
	3.5.5 Error Recovery and Diagnostics	39
4.	Semantic Constructor	42
5.	Possible Extensions	43
	5.1 Automatic Syntax Correction	43
	5.2 Parser Modification to Allow Simple Manipulation of the Parsing Stack by the Semantic Procedure	43

Page

6. Example Applications of Simple 45

6.1 Semantic Constructor. 45

6.2 A Command Language Meta System 49

6.2.1 The Model 49

6.2.2 The Table Generator 50

6.2.3 The Scanner. 54

6.2.4 Examples Using the Command Language Meta System 54

6.2.4.1 WYLBUR Example. 55

6.2.4.2 CRBE Example 57

Bibliography 59

Appendix A. 61

Appendix B. 62

Appendix C. 70

Appendix D. 75

Appendix E. 81

Appendix F. 85

Appendix G 89

LIST OF FIGURES

		Page
1.	Simple block diagram	2
2.	Example simple application	3
3.	Basic parsing algorithm	31
4.	Symbol recognition	36
5.	Flow chart for LOOK - the get next symbol procedure	37
6.	Parser flow chart	38
7.	Command language meta system - table generation	51
8.	Command language meta system - scanner	51

1. INTRODUCTION

SIMPLE is a specialized translator writing system designed to aid the implementation of an experimental graphic meta system in PL/I (George 1969 a & b). Although intended for writing preprocessors for PL/I, experience has demonstrated that these techniques can be used to implement various specialized languages (George 1967 a & b; George and Saal 1971).

SIMPLE is composed of three components: an executive, a syntax analyzer, and a semantic constructor as illustrated in Fig. 1.

The executive reads a block of data (i.e., variable initialization) and then passes control to the syntax analyzer and then to the semantic constructor.

The syntax analyzer reads the input syntax and constructs parsing tables which are then merged as data in a general skeleton parser, in source form (PL/I); this merged program is a specific parser for the language defined by the syntax and includes a parser, automatic error recovery and error diagnostics. The syntax analyzer has two output files: the specific parser, in source form (PL/I), and diagnostics related to the syntax.

The semantic constructor reads the semantics to be associated with the previous syntax and constructs a semantic procedure compatible with the specific parser; it also has diagnostic output for errors. The semantic constructor is defined using the syntax analyzer and a skeleton parser containing a short, hand-coded semantic procedure.

A language defined using SIMPLE functions is illustrated in Fig. 2. The input text is processed by the parser which calls the semantic procedure at appropriate times. The language processor has access to two output files: a source output and a diagnostic output. Both of these files are available to the parser and the

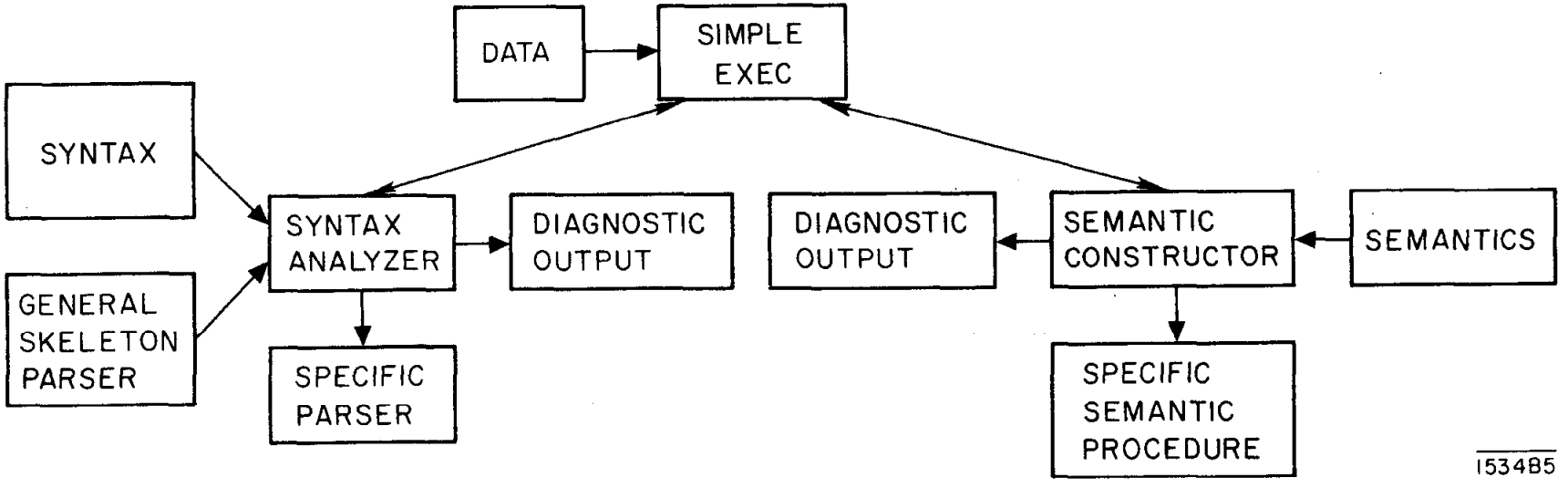


FIG. 1--Simple block diagram.

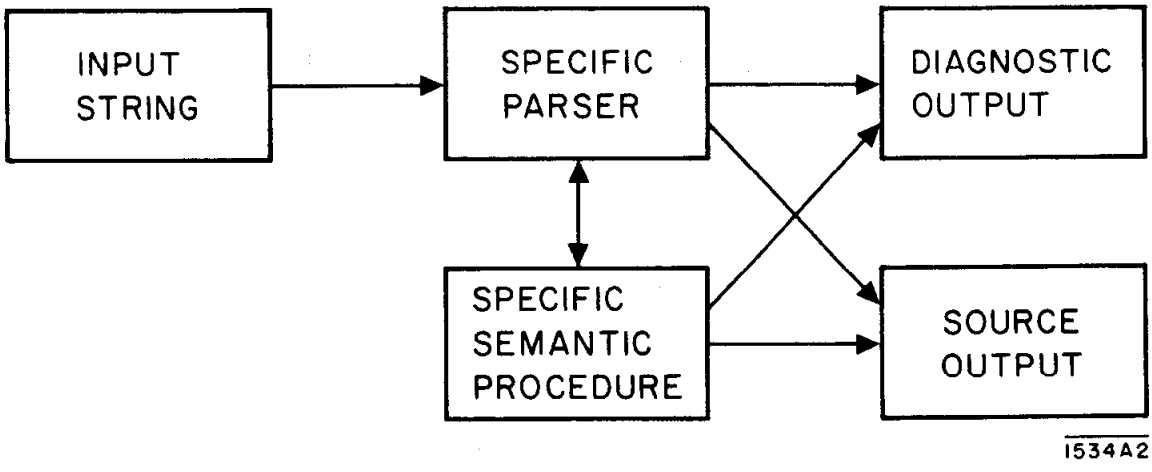


FIG. 2--Example simple application.

semantic procedure. A typical application would be to process input text and generate an equivalent source text (say PL/I) and error diagnostics, if any. The source output can then be compiled using a standard language processor.

2. INPUT DATA TO SIMPLE'S EXECUTIVE

The executive program initializes variables to be used by both the syntax analyzer and the semantic constructor. Any of these values may be changed by name value pairs appearing in the data file, SYNDATA (the data is read using the data directed input option in PL/1 and, hence, consists of the variable name, an "=" and the value as a legal constant in PL/1). The variables are:

NAME	TYPE	DEFAULT	EXPLANATION
ERRORSCAN	CHAR(20)VAR	*END*	That symbol in the syntax which is used in error recovery. When an error is detected when parsing, all current and future text until the first occurrence of this symbol is erased.
FILE1	CHAR(8)VAR	SYNTAX	Syntax equations input file.
FILE2	CHAR(8)VAR	SPARSER	Skeleton parser input file.
FILE3	CHAR(8)VAR	PARSER	Parsing program output file.
FILE4	CHAR(8)VAR	PSYNTAX	Syntax diagnostic output file.
FILE5	CHAR(8)VAR	SYNDATA	Input file for SIMPLE executive.
FILE6	CHAR(8)VAR	SEMANTICS	Semantic input file.
FILE7	CHAR(8)VAR	PSEMANT	Semantic diagnostic output file.
FILE8	CHAR(8)VAR	SEMANT	Semantic program output file.
INTEGER	CHAR(20)VAR	INTEGER	That symbol used in the syntax for an integer.
MLIM	FIXED BIN	20	Maximum number of symbols in the syntax.

NAME	TYPE	DEFAULT	EXPLANATION
MMLIM	FIXED BIN	20	Maximum number of non-basic symbols in the syntax.
NLIM	FIXED BIN	20	Maximum number of productions in the syntax.
PARSER_NAME	CHAR(8)	SEMANT	Name to be substituted for *PARSER* in FILE2; the procedure name for the parser procedure.
QUOTES	CHAR(20)VAR	"	That symbol used for quotes to force the STRING class.
RLIM	FIXED BIN	8	Maximum number of symbols on the right side in any production in the syntax.
SCAN_START	CHAR(20)VAR	*END*	That symbol not in the syntax which will restart the parsing.
SCAN_STOP	CHAR(20)VAR	*CODE*	That symbol in the syntax which, upon entry into the parsing stack, causes all input to be ignored by the parser until the symbol after SCAN_START.
SEMANT_NAME	CHAR(8)	CODE_OUT	Name to be substituted for *SEMANT* in FILE2; the name of the semantic procedure to be called by this parser.
SEND	CHAR(20)VAR	*END-SYNTAX*	Terminator for syntax.
SEQUENCE	CHAR(20)VAR	SEMANTICS	The initial symbol of the syntax; when it occurs in the stack, the parsing is terminated.
SINIT	CHAR(20)VAR	*SYNTAX*	Initiator for syntax analyzer.
SSEMANT	CHAR(20)VAR	*NO-SEMANT*	Indicates no semantics for this production.

NAME	TYPE	DEFAULT	EXPLANATION
SSEP	CHAR(20)VAR	*::=*	Separator for left-right sides.
STERM	CHAR(20)VAR	*;*	Terminator for syntax equations.
STRING	CHAR(20)VAR	STRING	That symbol in the syntax used for the string class.
SYM(*)	CHAR(20)VAR	SYM(1)='SEMANT' SYM(2)='CODA' SYM(3)='INTERPRE-TATIONS' SYM(4...20)=' '	Used for error recovery; those symbols which are expected to reside in the ith position of the parsing stack.
TERMINAL	CHAR(20)VAR	*END-SEMANTICS*	That symbol used to force the parsing to be completed.
WORD	CHAR(20)VAR	WORD	That symbol used in the syntax for the WORD class.

A listing of the executive is given in Appendix A.

3. SYNTAX ANALYZER AND PARSER

A simple precedence syntax analyzer was chosen for its simplicity, power and availability in a form suitable for modification. The basic analyzer was translated to PL/1 from an ALGOL listing obtained from N. Wirth (Wirth and Weber, 1966 a & b). Many sections were modified to take advantage of features of PL/1. The changes to the analyzer are:

1. The input section was modified to be free field and to mark productions with no semantics;
2. Maximum number of right part elements is variable;
3. Three terminal classes are recognized rather than two (this holds in the parser also);
4. The output section inserts PL/1 declarations into a skeleton parser rather than punching tables.

A complete listing of the syntax analyzer is given in Appendix B.

The skeleton parser is also a translation of an ALGOL parser (Wirth and Weber, 1966a, Shaw 1966) with the following modifications:

1. The parser uses precedence tables rather than precedence functions;
2. Three terminal classes are recognized rather than two;
3. An additional input scanner allows direct code emission independent of the parsing section;
4. Error recovery and diagnostics are provided and related to the grammar;
5. The semantic procedure is not called for those productions with no semantics.

Thus the output of the analyzer is a PL/1 program containing the parsing tables, error recovery and diagnostics; a listing of the skeleton parser is given in Appendix C.

3.1 Definitions and Notation

The formal definitions are included here for completeness (Wirth and Weber 1966a, Shaw 1966, Feldman and Gries 1967).

Upper case letters, special characters (*, + ...) or a string of these enclosed by < and > represent symbols.

Lower case letters represent strings of symbols.

Script letters represent sets.

An individual statement of the syntax is called a production and has a left side and a right side separated by '::='.

Terminal or basic symbols are those which appear only in right sides.

Nonterminal or nonbasic symbols are those which occur in left sides.

A grammar is a set of productions.

A grammar is a simple precedence grammar if:

1. The productions contain exactly one nonterminal symbol which appears only as a left side (i.e., the goal);
2. All left sides are single nonterminal symbols;
3. The productions contain a nonempty set of terminal symbols;
4. No two right sides of any pair of productions are identical;
5. Between any two symbols of the grammar one and only one of the precedence relations ($\lessdot$, =, > or no relation) holds.

The precedence relations are defined by:

1. A = B iff there is a production of the form U::= xABy in the grammar;
2. A < B iff there is a production of the form U::= xAVy and B$\epsilon\mathscr{L}$(V);

3. $A > B$ iff either

there is a production of the form

$U ::= xVBy$ and $A \in \mathscr{R}(V)$, or

there is a production of the form

$U ::= xVWy$ and $A \in \mathscr{R}(V)$ and $B \in \mathscr{L}(W)$,

where,

$\mathscr{L}(U) = \{S \mid \exists z(U ::= Sz)$ or $(\exists z(U ::= Vz)$

and $S \in \mathscr{L}(V))$

$\mathscr{R}(U) = \{S \ \exists z(U ::= zS)$ or

$(\exists z(U ::= zV)$ and $S \in \mathscr{R}(V))$

where z may be the null string.

3.2 Transforming a Grammar to Simple Precedence

In many cases, the grammar for a given language must be manipulated before it is a simple precedence grammar. The problem areas are the requirement for unique precedence relations between any two symbols of the language and that no two productions have identical right sides. Within the literature, many formal properties about precedence languages are discussed and each uses his own definitions. For presenting these results, the definition of a simple precedence grammar is given in Section 3.1 and S-precedence is defined by:

Simple precedence ≡ S-precedence plus unique right sides

Some of the formal properties are:

1. Wirth and Weber's parsing algorithm yields a unique canonical parse for any sentence of any simple precedence language (Wirth and Weber 1966a, Shaw 1966).
2. A context free grammar can be transformed to a simple precedence grammar but the terminal language may be altered (Presser 1968; Gray 1969; Presser and Melkanoff 1969).

3. Any context free grammar can be transformed to a S-precedence grammar, and there is no deterministic parsing algorithm for all S-precedence grammars (Fischer 1969). The transformation proof requires Chomsky normal form of a grammar and is not useful as a practical algorithm.
4. Any context free grammar can be transformed to a S-precedence grammar without modification of the terminal language (Learner and Lim 1970; McAfee and Presser 1970). These proofs are different but both are directly useful as practical techniques.
5. Any context free grammar with unique right sides can be transformed into a S-precedence grammar with at most two duplications of any right side of any production (Learner and Lim 1970).

I had also studied these transformations using methods similar to Learner and Lim's but was unable to complete the formal proof (George 1969c). The proof is short with the proper form but does not lead to a practical algorithm (Fischer 1969); Learner and Lim's approach results in a more difficult proof but yields a practically useful algorithm; it is also intuitively easier to understand.

3.2.1 Removing Precedence Conflicts

Precedence conflicts* can be removed by several means, however the method presented here will be restricted such that it does not cause a change in the terminal language or require a change in the associated semantics of any production of the grammar. The changes of interest are those which could be incorporated in the syntax analyzer of SIMPLE and be invisible to a user.

*A precedence conflict means that more than one of the precedence relations holds between two symbols of the grammar.

From the formal work, this can not always be accomplished for an arbitrary context free grammar, but if the terminal language is altered or the associated semantics must be modified, then the user must make these changes before SIMPLE can be utilized. However, many times the changes required are significant and the user should be conscious of them.

The techniques presented are intended to be intuitive and easy to understand.

An artificial production is a production with no associated semantics and only one element on the right side (Shaw 1966, p. 145; also called an intermediate production, Feldman and Gries 1967, p.28).

A left restricted expansion (LRE) of A replaces A in the right sides of all productions, except where it is the left-most symbol, by the same new non-terminal A_i and adds the artificial production A_i : : = A to the grammar (Learner and Lim 1970).

A right restricted expansion (RRE) of A replaces A in the right sides of all productions, except where it is the right most symbol, by the same new non-terminal A_i and adds the artificial production A_i : : = A to the grammar (Learner and Lim 1970).

Lemma 1: The precedence relation = between two symbols A and B (i.e. A = B) can be changed to < by a left restricted expansion of B.

Proof: Let A = B, then productions of the form

U : : = x A B y exist

By the LRE these become

U : : = x A B_1 y

and B_1 : : = B is added to the grammar

Thus, A = B_1 and A < B.

Lemma 2: The precedence relation = between two symbols A and B (i.e. A = B) can be changed to > by a right restricted expansion of A.

Proof: Let A = B, then productions of the form

U ::= x A B y exist

By the RRE these become

U ::= x A_1 B y

and A_1 ::= A is added to the grammar

Thus, A_1 = B and A > B.

Lemma 3: The precedence relation < between two symbols A and B (i.e. A < B) can be changed to > by a right restricted expansion of A.

Proof: Let A < B, then there exist productions of the form

U ::= x A V y and B ϵ $\mathscr{L}$ (V)

By the RRE these become

U ::= x A_1 V y and B ϵ $\mathscr{L}$(V)

and A_1 ::= A is added to the grammar

Thus A_1 = V, A_1 < B and A > B.

These lemmas provide the techniques for removing precedence conflicts between two symbols; the changes in the grammar do not affect the terminal language or the associated semantics. The precedence conflicts which can occur between any two symbols are (=, <), (=, >), (< , >) and (=,< , >).

Th 1: The precedence violation pair (=, <) between two symbols A and B (i.e. A = B and A < B) can be removed by a left restricted expansion of B (i.e. change the = to < by Lemma 1); however, new violations may be introduced.

Proof: Lemma 1 for removal of the original conflict.

No left sets are altered by the expansion and some right sets may have the new symbol B_i included, hence relationships between symbols other than A, B or B_i are unchanged. The only symbols whose relationship may cause violations are those adjacent to a B in the original grammar.

Let the symbol Z occur to the left of B and Y to the right of B in the original grammar, then

Orig. relation	new relation (after LRE)
$Z = B$	$Z = B_i \quad Z < B$
$Z < B$	$Z < B$
$Z > B$	$Z > B \quad Z > B_i$ (possible)
$B = Y$	$B = Y$ or $B_i = Y$ & $B > Y$
$B < Y$	$B < Y$ or $B_i < Y$ & $B > Y$
$B > Y$	$B > Y$

Thus, the conflicts which could be introduced are

B (=, >) Y from productions of the form

U ::= B Y d and W ::= e V B Y f

and

B (<, >) Y from productions of the form

U ::= B T d and $Y \in \mathscr{L}(T)$

W ::= e V B T f and $Y \in \mathscr{L}(T)$

One might consider removing the violation pair (=, <) by applying a right restricted expansion to A (i.e. changing the = to > by Lemma 2 and the < to > by Lemma 3); however, this leaves the original violation pair between A_i and B.

Th 2: The precedence violation pair (=, >) between two symbols A and B (i.e. A = B and A > B) can be removed by a right restricted expansion of A (i.e. change the = to > by Lemma 2); however, new violations can be introduced.

Proof: *Lemma 2 for removal of the original conflict*

No right sets are altered by the expansion and some left sets may have the new symbol A_i included, hence relationships between symbols other than A, B or A_i are unchanged. The only sumbols whose relationships may cause violations are those adjacent to an A in the original grammar.

Let the symbol Z occur to the left of A and Y to the right of A in the original grammar, then

orig. relation	new relation
$A = Y$	$A_i = Y \quad A > Y$
$A < Y$	$A_i < Y \quad A > Y$
$A > Y$	$A_i < Y$ or $A_i = Y$ and $A > Y$
$Z = A$	$Z = A$ or $Z = A_i$ & $Z < A$
$Z < A$	$Z < A \quad Z < A_i$ (possible)
$Z > A$	$Z > A \quad Z > A_i$ (possible)

Thus the conflict (=, <) could be introduced between Z and A from original productions of the form

U ::= d Z A and W ::= e Z A V f.

Th 3: The precedence violation pair (< , >) between two symbols A and B (i.e. A < B and A > B) can be removed by a right restricted expansion of A (i.e. change the < to > by Lemma 3); however, new violations can be introduced.

Proof: Lemma 3 for removal of the original violation.

Second part of Theorem 2 for rest.

Th 4: The precedence violation triple (=, < , >) between two symbols A and B (i.e. A = B , A < B and A > B) can be removed by a right restricted expansion of A (i.e. change the = to > by Lemma 2 and the < to > by Lemma 3); however, new violations can be introduced.

Proof: Lemmas 2 and 3 for removal of the original conflict.

Second part of Theorem 2 for rest.

Th 5: A context free grammar with unique right sides can be transformed to a S-precedence grammar with at most two duplications of any right side.

Proof: Find all the violations between two symbols A and B.

Case 1: A (=, <) B

Theorem 1 substitutes B_1 ::= B and the only B's remaining are

B_1 ::= B , and

U ::= B y where y may be null

The only violation which can be introduced is one between B and C, where C occurs immediately to right of B in some production.

Case A: B (=, >) C

Theorem 2 adds B_2 ::= B and changes U ::= B V y to U ::= B_2 V y.

Thus, the only B's remaining are

B_1 ::= B (Th 1)

B_2 ::= B (Th 2)

U ::= B (from original grammar)

The only violations from Theorem 2 involve symbols immediately to the left of a B after applying Theorem 1, of which there are none. Therefore, after two levels no new violations will be

introduced. Further, for an expansion to be required for B, B would have to occur adjacent to some symbol to generate some precedence conflict; since it does not, only two duplications are possible.

Case B: B (< , >) C

Theorem 3 adds B_2 ::= B and becomes same as Case A.

Case 2: A (=, >) B

Theorem 2 leaves the following productions with A's

A_1 ::= A , and

U ::= y A

The only violations which can be introduced is one between A and C where C occurs immediately to left of A.

C (=, <) A

Theorem 1 adds A_2 ::= A and changes U ::= y A to U ::= y A_2

Thus the only A's remaining are

A_1 ::= A (Th 2)

A_2 ::= A (Th 1)

U ::= A (from the original grammar).

By Theorem 1, the only new violations which can be introduced must occur with a symbol immediately to the right of an A after the application of Theorem 2; since there are no symbols of this type, no new violations will be introduced by Theorem 1. Further, for an expansion of A to be required, A would have to occur adjacent to some symbol to generate some precedence conflict; since it does not, only two duplications result.

Case 3: A (< , >) B

Theorem 3, then same as Case 2.

Case 4: A (=, < , >) B

Theorem 4, then same as Case 2.

Learner and Lim's algorithm is recursive, but since the grammars are finite, the number of duplicates of right hand sides is at most two, I suspect that the algorithm does not need to be recursive, but perhaps related to the total number of symbols of the original grammar.

3.2.2 Transforming a S-Precedence Grammar to Simple Precedence

Section 3.2.1 shows how to transform any context free grammar to a S-precedence grammar. If the transformed grammar is only S-precedence, it must be transformed to simple precedence before being useable within SIMPLE. Generally, this requires a change in the terminal language or splitting of productions and the corresponding change in the associated semantics. These changes must be specified by the user and an example is given in the next section.

3.2.3 Transformation Examples

1. Violation pair (=, <) (Shaw 1966, example 4 pp. 139-141).

S : : = E

E : : = E + T

E : : = T

T : : = T * F

T : : = F

F : : = (E)

F : : = <VAR>

The violations are + = <T and (= <E.

For + and T,

+ = T results from E::= E+T

+ < T results from E::= E+T and T$\epsilon\mathscr{L}$(T)

Using Th 1, change T to <TT> resulting in the grammar;

S::= E

E::= E + <TT>

E::= <TT>

<TT>::= T

T::= T * F

T::= F

F::=(E)

F::= <VAR>

This removes the violation pair (=, <) between + and T, but not the pair for (and E.

For (and E,

(= E results from F::= (E)

(< E results from F::= (E) and E$\epsilon\mathscr{L}$(E)

Using Th 1, change E to <EE> resulting in the grammar;

S::= <EE>

<EE>::= E

E::= E + <TT>

E::= <TT>

<TT>::= T

T::= T * F

T::= F

T: := (< EE >)

F: := < VAR >

Which is a simple precedence grammar.

2. Violation pair (= , >).

Consider the above grammar modified to be right recursive instead of left recursive.

S: := E

E: := T + E

E: := T

T: := F * T

T: := F

F: := (E)

F: := < VAR >

the violations are T = > + and E = >).

For the T and +, use Th 2 and change T to < TT >; for the E and), use Th 2 and change E to < EE >, resulting in the grammar:

S: := < EE >

< EE >: := E

E: := < TT > + E

E: := < TT >

< TT >: := T

T: := F * T

T: := F

F: := (< EE >)

F: := < VAR >

Now the grammar is a simple precedence grammar.

3. Violation pair (<, >).

Consider the grammar:

N: := R

N: := S

R: := W A T X

S: := Y U B Z

T: := B

U: := M A

The violation is A >< B.

A < B results from R: := WATX and B$\epsilon\mathscr{L}$(T)

A > B results from S: := YUBZ and A$\epsilon\mathscr{R}$(U).

Using Th 3, change A to C resulting in the grammar:

N: := R

N: := S

C: := A

R: := W C T X

S: := Y U B Z

T: := B

U: := M A

Which is a simple precedence grammar. Note that the A in U: := MA was not changed.

Consider the grammar:

N: := R

N: := S

R: := W A T X

S: := Y U V Z

T: := B

U: := M A

V: := B K

The violation is A > < B.

A < B results from R: := WATX and B$\epsilon\mathscr{L}$(T)

A > B results from S: := YUVZ and A$\epsilon\mathscr{R}$(U) and B$\epsilon\mathscr{L}$(V)

Using Th 3, change A to C resulting in the grammar:

N: := R

N: := S

C: := A

R: := W C T X

S: := Y U V Z

T: := B

U: := M A

V: := B K

Which is a simple precedence grammar. Note that the A in U: := MA was not changed.

4. Consider the syntax for simple assignment statement.

<STAT> : := <VAR> <:=> <EXPR>

<EXPR> : := <EXPR> + <TERM>

<EXPR> : := <EXPR> - <TERM>

<EXPR> : := - <TERM>

<EXPR> : := <TERM>

<TERM> : := <TERM> × <FACTOR>

<TERM> : := <TERM> / <FACTOR>

<TERM> : := <FACTOR>

<FACTOR>::=<FACTOR>*<PRIMARY>

<FACTOR>::=<PRIMARY>

<PRIMARY>::=(<EXPR>)

<PRIMARY>::=<VAR>

<PRIMARY>::=<NUMBER>

The violations are:

<:=>	=<	<EXPR>
(	=<	<EXPR>
+	=<	<TERM>
-	=<	<TERM> two cases
×	=<	<FACTOR>
/	=<	<FACTOR>

This example suggests that the symbols which have been replaced must be recorded to prevent future redundant substitutions.

Using Th 1 repeatedly, the grammar becomes:

<STAT>::=<VAR><:=><EXPRA>

<EXPRA>::=<EXPR>

<EXPR>::=<EXPR>+<TERMA>

<EXPR>::=<EXPR>-<TERMA>

<EXPR>::=-<TERMA>

<EXPR>::=<TERMA>

<TERMA>::=<TERM>

<TERM>::=<TERM>×<FACTORA>

<TERM>::=<TERM>/<FACTORA>

<TERM>::=<FACTORA>

<FACTORA>::=<FACTOR>

<FACTOR> ::= <FACTOR> * <PRIMARY>

<FACTOR> ::= <PRIMARY>

<PRIMARY> ::= (<EXPRA>)

<PRIMARY> ::= <VAR>

<PRIMARY> ::= <NUMBER>

which is a simple precedence grammar.

5. Consider an early version of the syntax for SPIRES, an information retrieval system (George 1967b; Parker 1967).

<SEARCH> ::= <FIND> <REQLIST> ; <END>

<REQLIST> ::= <COMPSEARCH>

<REQLIST> ::= <REQLIST> ; <COMPSEARCH>

<REQLIST> ::= <REQLIST> ; <OR> <COMPSEARCH>

<COMPSEARCH> ::= <FACTOR>

<COMPSEARCH> ::= <COMPSEARCH> <OR> <FACTOR>

<FACTOR> ::= <SIMPSEARCH>

<FACTOR> ::= <FACTOR> <AND> <SIMPSEARCH>

<PHRASE> ::= <WORD>

<PHRASE> ::= <PHRASE> <WORD>

<SIMPSEARCH> ::= (<COMPSEARCH>)

<SIMPSEARCH> ::= <AUTHOR> <PHRASE>

<SIMPSEARCH> ::= <DATE> <BETWEEN> <PHRASE>
<AND> <PHRASE>

The violations are:

a. <FIND> = < <REQLIST>

b. ; = < <COMPSEARCH>

c. (= < <COMPSEARCH>

d. <OR> = < <COMPSEARCH>

e. <OR> = < <FACTOR>

f. <AUTHOR> = < <PHRASE>

g. <BETWEEN> = < <PHRASE>

h. <AND> = < <PHRASE>

i. <PHRASE> = < <AND>

a. Changing <REQLIST> to <REQLIST-> as specified by Th 1 will result in the violation pair (=, >) between <REQLIST> and ";" as discussed in the theorem. This is an error which requires a production to be split.

b, c and d. Using Th 1, change <COMPSEARCH> to <COMPSEARCH->.

e. Using Th 1, change <FACTOR> to <FACTOR->.

Thus, two productions with a right side of <FACTOR> result; the solution is a different terminal symbol for one of the <OR>'s.

f, g and h. Using Th 1, change <PHRASE> to <PHRASE->.

i. Using Th 2, change only one <PHRASE-> to <PHRASE+>.

If the correction for i is made before f, g and h, then the steps would be:

Change <PHRASE> after <BETWEEN> to <PHRASE+> and add <PHRASE+>::= <PHRASE> ; do not change other <PHRASE>'s since this would remove <PHRASE> ϵ R(<FACTOR>) as specified in the theorem.

Change <PHRASE> after <AUTHOR> to <PHRASE->,

<PHRASE+>::=<PHRASE> to <PHRASE+>::=<PHRASE-> and all of the <PHRASE>'s to <PHRASE-> except those where <PHRASE> is the left side.

The corrected grammar is:

<SEARCH>::=<FIND><REQLIST-><END>*

<REQLIST->::=<REQLIST>;

<REQLIST>::=<COMPSEARCH->

<REQLIST>::=<REQLIST>;<COMPSEARCH->

<REQLIST>::=<REQLIST>;<ORA><COMPSEARCH->**

<COMPSEARCH->::=<COMPSEARCH>

<COMPSEARCH>::=<FACTOR->

<COMPSEARCH>::=<COMPSEARCH><OR><FACTOR->

<FACTOR->::=<FACTOR>

<FACTOR>::=<SIMPSEARCH>

<FACTOR>::=<FACTOR><AND><SIMPSEARCH>

<PHRASE+>::=<PHRASE->

<PHRASE->::=<PHRASE>

<PHRASE>::=<WORD>

<PHRASE>::=<PHRASE><WORD>

<SIMPSEARCH>::=(<COMPSEARCH->)

<SIMPSEARCH>::=<AUTHOR><PHRASE->

<SIMPSEARCH>::=<DATE><BETWEEN><PHRASE+>
<AND><PHRASE->

---

*This production was split.

**This <OR> was changed.

3.3 Input Conventions for the Syntax Analyzer

The input for the syntax analyzer (i.e., the productions) is contained in a file whose default name is SYNTAX (setting this name is explained in Section 2). The formal definition of the syntax is:

<SYNTAX> ::= <SINIT> <PRODUCTIONS> <SEND>

<PRODUCTIONS> ::= <PRODUCTION>

::= <PRODUCTIONS> <STERM> <PRODUCTION>

<PRODUCTION> ::= <LEFT-PART> <SSEP> <RIGHT-PART>

::= <LEFT-PART> <SSEP> <RIGHT-PART> <SSEMANT>

<LEFT-PART> ::= <SYMBOL>

<RIGHT-PART> ::= <SYMBOL>

::= <RIGHT-PART> <SYMBOL>

<SYMBOL> ::= any string excluding blanks

The default values are:

<SINIT> = *SYNTAX*

<SEND> = *END-SYNTAX*

<STERM> = *;*

<SSEP> = *::=*

<SSEMANT> = *NO-SEMANT*

The input is free field card images using blanks or a new card to separate symbols; only the first 20 characters of a symbol are used.

In actual use there are two additional limits:

1. Upper limit on number of productions;
2. Upper limit on number of symbols in any right part.

If more productions than the limit of productions are used, then those productions between the limit less one and the last productions are lost; similarly, for more

symbols in the right part than the limit. Note that both of these are input parameters to SIMPLE (Section 2).

If the left part has more than one symbol then the last symbol in the left part is used.

3.4 Syntax Analyzer Output

In addition to inserting the necessary declarations and initialization into the skeleton parser, the syntax analyzer generates a file (FILE4 whose default name is PSYNTAX) which contains information about the syntax and any errors. This output consists of:

1. Productions — The productions are numbered in the order that they are read in and this number is used to select the applicable portion of the semantic procedure.
2. Basic and nonbasic symbols — The basic and nonbasic symbols are assigned a unique number.
3. KEY and PRTB tables (Shaw 1966a p. 194) — These are used by the parser in determining the production number and the left part of the production of a reducible substring. "KEY(i) represents for the $i^{\underline{th}}$ symbol (i corresponds to the number assigned in 2) the index in the production table PRTB, where those productions are listed whose right part string begins with the $i^{\underline{th}}$ symbol. For each production, the right part is listed without its leftmost symbol, followed by the production number (negative) and the left part symbol of the production. The end of the list of productions referenced via KEY(i) is marked with a 0 entry in PRTB." If a production has no semantics then the production number in PRTB is adjusted to be out of range (by the number of productions).
4. Right and left symbol sets — These are sometimes useful in removing conflicts.

5. PRECEDENCE Matrix — Two symbols x and y are related (either x=y, x<y, x>y or no relation) by the entry in the $i^{\underline{th}}$ row (where i is the number corresponding to x) and $j^{\underline{th}}$ column (j corresponding to y) of the matrix.

6. DIAGNOSTICS

   a. For a correct syntax

      NO PRECEDENCE VIOLATIONS OCCURRED

   b. For an incorrect syntax

      1. PRECEDENCE VIOLATIONS OCCURRED

         HINTS REGARDING PRECEDENCE VIOLATION

         The most recent production number which causes a violation followed by the two symbols separated by the two relations.

   c. Incorrect input file

      ***** ENDFILE SYNTAX INPUT - NO

      followed by the value of SEND (Section 2).

      SEND missing generally causes no problems. If there is no additional syntax output, then the symbol SINIT was never encountered (Section 2).

3.5 Parser

One of the principal advantages of the simple precedence system is the parser, which, for a correct syntax, yields a unique canonical parse with no backtracking (Wirth and Weber 1966a; Shaw 1966). This permits the syntactical analysis (parsing) to be separated from the semantics; this is both a blessing and a headache.

The advantage of this separation is that the parser can be protected from interference (or modification) from the associated semantics. This protection is very important when a complete parser is supplied to any user; it limits debugging faults and permits confident use without a detailed knowledge of the internal methods.

However, this separation also limits the power of the applications. Namely, no semantic process can alter or change the parsing (i.e., the system is entirely syntactically driven); this sometimes results in an awkward syntax or may not be applicable to a class of desirable languages. Section 5.2 discusses this further and illustrates an extension which relaxes this requirement, still preserving an acceptable level of protection.

The parsing algorithm depends upon the precedence relations <, = and > (Wirth and Weber 1966a; Shaw 1966) according to:

1. The relation = holds between all adjacent symbols within a symbol which is directly reducible;
2. The relation < holds between the symbol immediately preceding a reducible string and the leftmost symbol of that string;
3. The relation > holds between the rightmost symbol and the symbol immediately following that string.

The basic parsing algorithm consists of locating a string S_j ---- S_k such that $S_\ell = S_{\ell+1}$ for ℓ=j, j+1, --- k-1 and $S_{j-1} < S_j$ and $S_k > S_{k+1}$. This string S_j --- S_k is then a reducible substring and corresponds to some production $U ::= S_j --- S_k$. The semantics for the production may then be performed and then the string $S_j --- S_k$ is replaced by the left side of the production. This is illustrated in Fig. 3.

The parser consists of five parts:

1. Declarations in the parser;
2. Declarations and initialization inserted by the syntax analyzer (i.e., dependent upon the grammar);
3. Symbol recognition;
4. Parsing;
5. Error recovery.

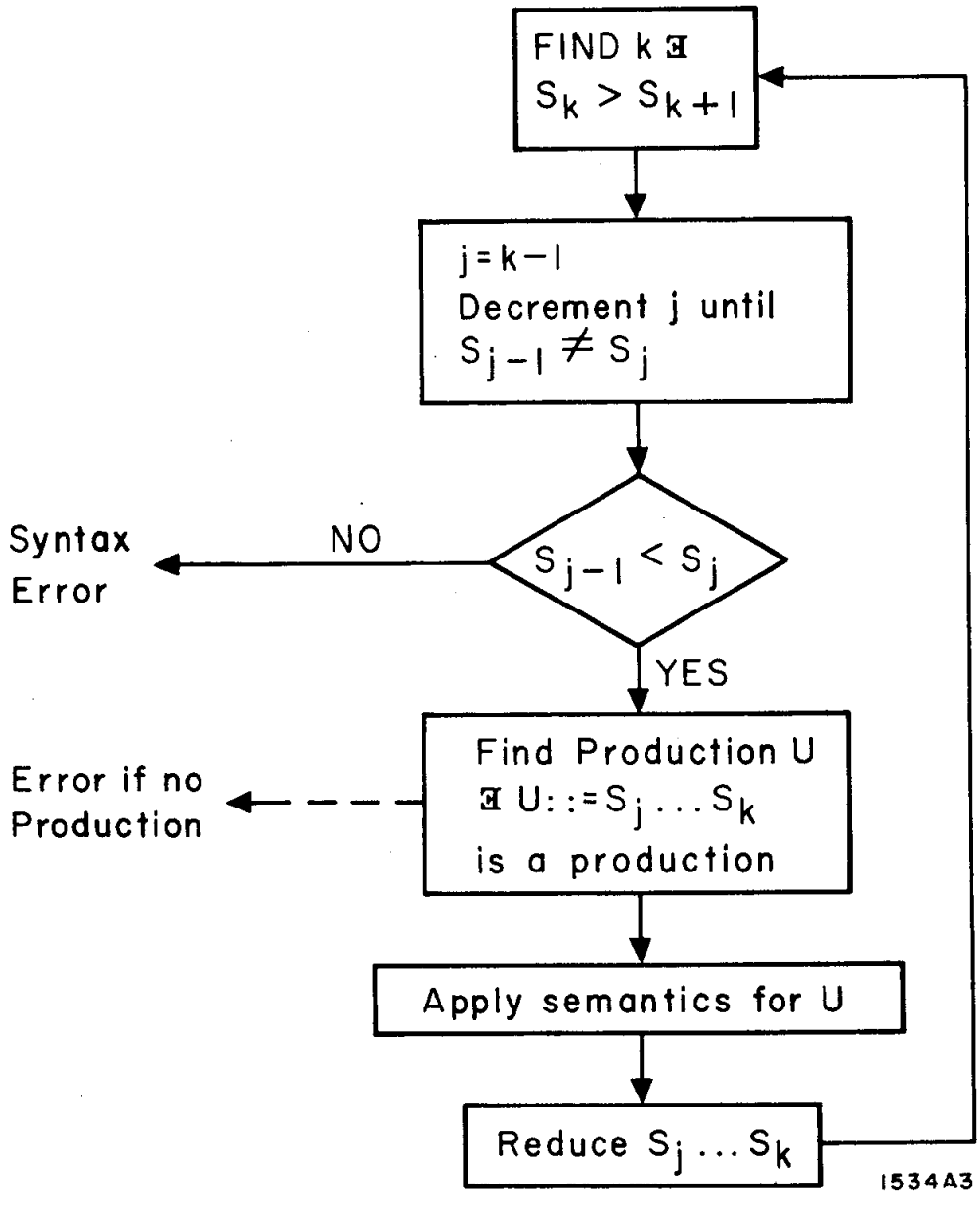


FIG. 3--Basic parsing algorithm.

3.5.1 Declarations in the Parser

These declarations reside in the parser since they are related to the parsing technique and not to the individual grammar.

NAME	TYPE	VALUE	EXPLANATION
ANS	FIXED BIN	initially 0	For use in the semantic routine
ERROR	BIT(1)	initially '0'B	For use in the semantic routine to indicate an error; upon return to the parser, if ERROR true ('1'B) then parsing is terminated.
J	FIXED BIN	--	Left hand stack pointer; copy of it passed to semantic routine.
K	FIXED BIN	--	Right hand stack pointer; copy is passed to the semantic routine.
INPUT	CHAR(100)VAR	--	Input string buffer.
INPUT	CHAR(7)VAR	SOURCE	Input file identified as //GO.SOURCE. Contains the input to be parsed.
OUTPUT	CHAR(7)VAR	OUTPUT	Output file identified as //GO.OUTPUT.
POUT	CHAR(7)VAR	DIAG	Diagnostic output file identified as //GO.DIAG.
SYM	FIXED BIN	--	Numerical form of the current input symbol.
SYMS	CHAR(400)VAR	--	String form of the current input symbol.
S(0:50)	FIXED BIN	initially set to 0	Parsing stack (numerical form)
V(0:50)	CHAR(400)VAR	initially null	Associated value stack to the parsing stack.

3.5.2 Declarations and Initialization Inserted by the Syntax Analyzer

The declarations and values for these variables are inserted by the syntax analyzer since they are determined by the grammar.

NAME	TYPE	EXPLANATION
BASSYM(*)	CHAR(20)VAR	Contains the basic symbols of the grammar with the three types WORD, INTEGER and STRING removed and the value of TERMINAL added.
BASVAL(*)	FIXED BIN	The associated numerical form of BASSYM.
ERRORSCAN	CHAR(20)VAR	Termination symbol for error recovery.
H(0:*, 0:*)	CHAR(1)	The precedence matrix; each entry is =, <, > or blank.
HINITIAL	Procedure	This procedure is automatically called upon entry to the parser to initialize the matrix H. Within the procedure, the variable J contains triples indicating the nonblank entries in H; Row, Column, [0, 1, 2] where 0 means =, 1 means <, 2 means >. This solution was forced by the PL/I compiler due to maximum string length in the INITIAL statement.
HLIM	FIXED BIN	Upper limit for each dimension of the H matrix and KEY matrix.
KEY(0:*)	FIXED BIN	Index in PRTB for those productions whose right part string begins with the i<u>th</u> symbol.
M	FIXED BIN	DIMENSION of BASSYM and BASVAL
N	FIXED BIN	Number of productions
PRTB(0:*)	FIXED BIN	Contains the productions without the left-most symbol of the right part and with the production number (negative) and the left part symbol of the production. Productions with the same leftmost symbol of the right part are together and these groups are separated by 0's.
QUOTES	CHAR(20)VAR	That symbol which turns on and off the string class recognition.
SCAN_START	CHAR(20)VAR	That symbol which terminates the alternate scanner and returns to the parsing section.
XINTEGER	FIXED BIN	Numerical form of the symbol in the grammar used for the integer class.

NAME	TYPE	EXPLANATION
XSCAN_STOP	FIXED BIN	Numerical form of the symbol in the syntax which activates the alternate scanner just before it is inserted into the parsing stack.
XSEQ	FIXED BIN	Numerical form of the goal. When this appears as the rightmost element of the parsing stack, the parsing is terminated and control is returned to the calling program.
XSTRING	FIXED BIN	Numerical form of the symbol in the grammar used for the string class.
XSYM(10)	FIXED BIN	Used for error recovery. (See error recovery section.)
XTERM	FIXED BIN	Numerical form of the symbol whose precedence is such that it will force all parsing to be completed and prevent scanning across the beginning of the parsing stack.
XWORD	FIXED BIN	Numerical form of the symbol in the grammar used for the word class.

3.5.3 Symbol Recognition

The function of the symbol recognizer is to scan the input file for the next syntactical unit and to assign this symbol the unique number originated by the syntax analyzer. The recognizer classifies all symbols into four classes:

1. INTEGER CLASS
2. WORD CLASS
3. STRING CLASS
4. RESERVED WORDS

The integer class is defined by:

INTEGER ::= DIGIT

::= INTEGER DIGIT

DIGIT ::= 0|1|2|3|4|5|6|7|8|9

The word class is any string of characters starting with a non-digit and excluding blanks, single character reserved words and QUOTES if it is a single character.

The string class is any string of characters including reserved words and surrounded by QUOTES; the string corresponding to QUOTES is erased.

Reserved words are those words contained in the BASSYM matrix.

The separators for the word class are blanks, a single character QUOTES and single character reserved words. The separators for the integer class are any non-digit character. The entire character string enclosed in QUOTES is recognized as a string as it appears; the QUOTES are removed since they are not part of the syntax. A flow chart of the symbol recognition is given in Figs. 4 and 5.

3.5.4 Parsing

The parser is a modification of the basic parsing algorithm given at the beginning of Section 3.5. The flow chart for the parser is given in Fig. 6. "S" is a stack which contains the partially reduced string at any time. The input string is copied one symbol at a time into SYM and SYMS. If the rightmost element of S is > SYM then S is scanned to the left from the current right end until $S_{i-1} \neq S_i$; at this point if $S_{i-1} < S_i$ then we are guaranteed (if the string is in the language) that there is a production whose right side is S_i --- S_j. We then perform a "semantic reduction" on the value stack V_i --- V_j (i.e., call the semantic procedure) and then reduce the string S_i --- S_j by replacing it by the left side of the corresponding production.

Input to the parser is in a file named SOURCE; the parser has two output files, one for diagnostics (internal name, POUT, external name DIAG) and one for semantic output (internal name OUTPUT, external name OUTPUT). Both output files are used by the parser and both may be used by the semantics.

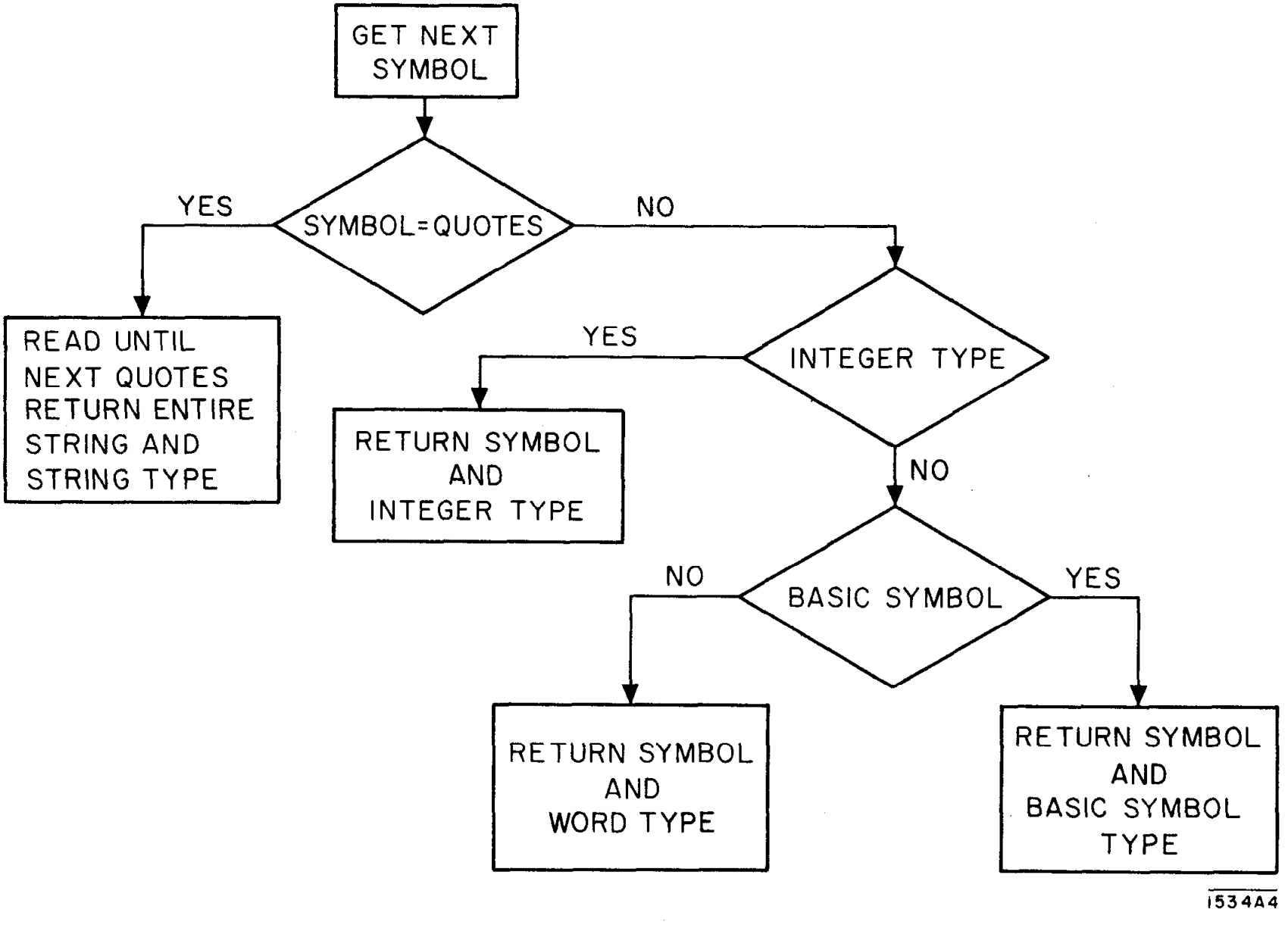


FIG. 4--Symbol recognition.

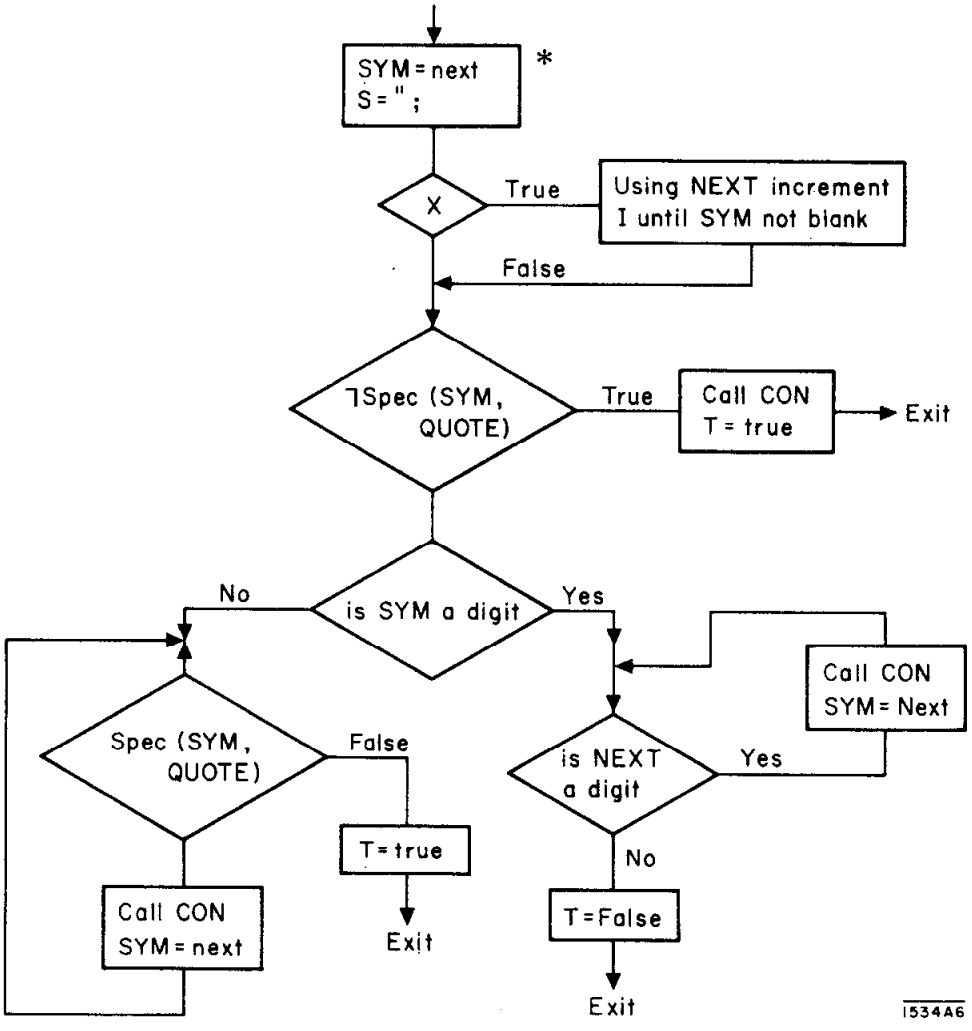
Entry Variables

S Output string

I Input character pointer

T T is set to False if integer else true

X if X true then Blanks removed



*SPEC returns true if first argument is not a separating character.
*CON concatenates SYM to end of S and increments I.
*NEXT returns the character pointed at by I in the input string. If I > length of input string, 80 more characters are read, a blank is concatenated to end and I is set to 1.

FIG. 5--Flow chart for LOOK — the get next symbol procedure.

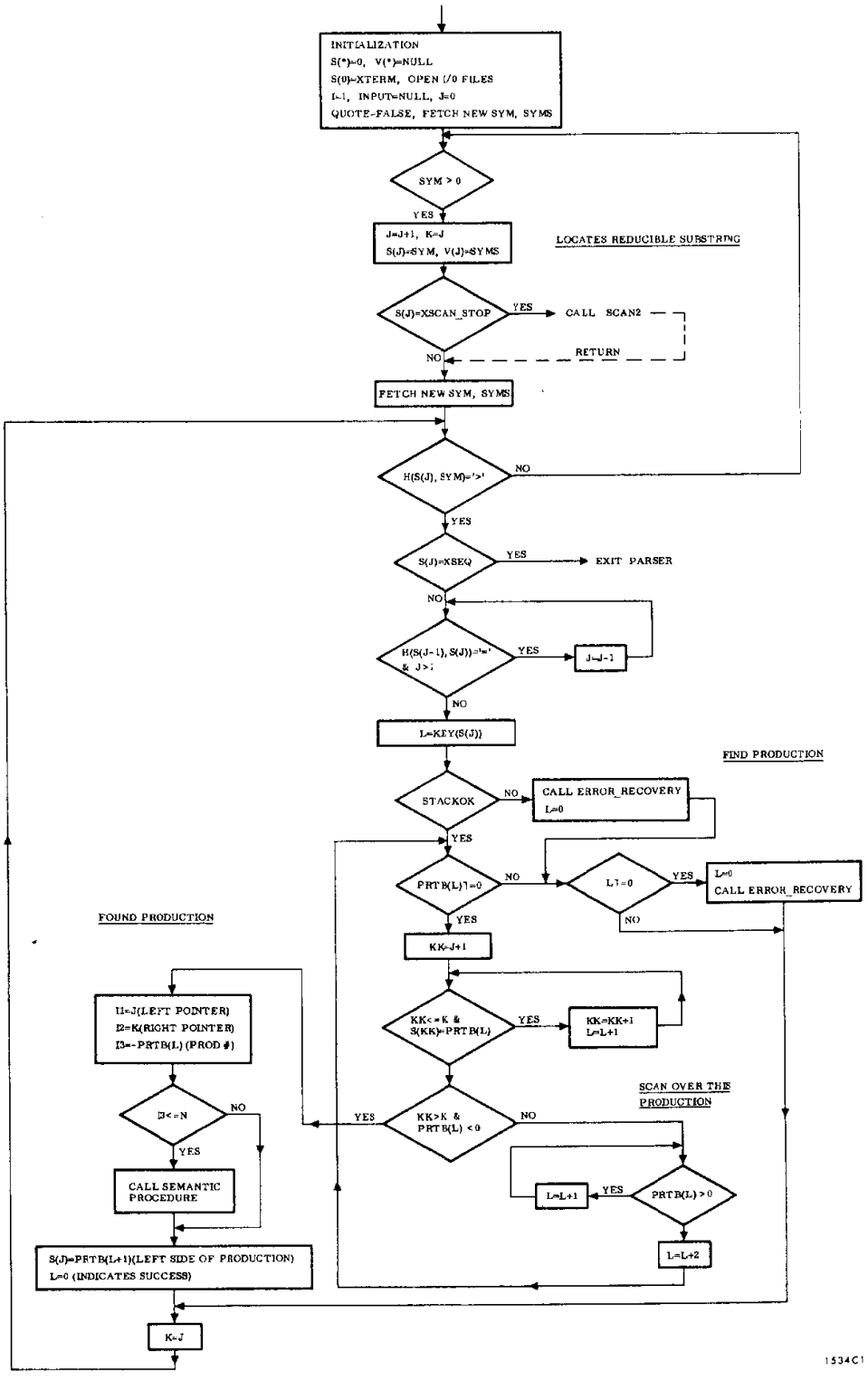


FIG. 6--Parser flow chart.

When the SCAN-STOP symbol is moved to the parsing stack, procedure SCAN2 is activated. SCAN2 simply reads the input and copies it to the output file (this is the only use of the OUTPUT file in the parser) until the SCAN-START symbol is detected (the SCAN-START symbol is effectively erased). This facility allows the mixture of special code and the normal output code within one input string.

3.5.5 Error Recovery and Diagnostics †

> "There has also been very little effort on the problems of automatic error detection and recovery in syntax-directed processors. Once again, even a bad system would be of great value to users." (Feldman and Gries 1967, p. 111)

After using the syntax for implementing several different languages (George 1967b, 1969b) a simple method for error recovery and useable automatic diagnostics has finally evolved. This has primarily resulted from careful analysis of the parsing stack and the classification of the input symbols.

With a simple precedence system, the earlier an error is detected (i.e., with the least amount of parsing) the easier it is to recover and issue meaningful diagnostics. Precedence functions were utilized in an earlier system (Wirth and Weber 1966a, b) and led to complications for error detection. With the precedence functions, the blank relation is effectively removed and several steps of parsing can occur before an error is detected; in fact, the only type of error to be detected is an illegal production (i.e., no production matches the string to be reduced). The problem of restoring the parsing stack after several illegal reductions is complex; further, one cannot automatically restore the actions performed by the associated illegal semantic activations. Also, automatic diagnostics were impossible since the blank entries were missing.

---

†Leinius (Leinius 1970) analyzes and classifies syntax errors in simple precedence and LR(K) languages. He developes general techniques for detecting errors (equivalent to the detection methods used here) and specifying, syntactically, error recovery for any language of these classes. His techniques are more general than those presented here, but are not needed in simple languages. The techniques presented here have proven adequate for applications involving simple languages.

The solution was to try to detect syntax errors as soon as possible and keep the blank entries for diagnostic purposes. With a change of Wirth and Weber's parsing algorithm, the errors can be detected earlier (i.e., use the precedence matrix and not the functions). When searching for a reducible substring, the search is only started when a '>' relation exists between the rightmost symbol of the stack and the next symbol. A scan is then initiated to scan to the left in the stack while the '=' relation holds between adjacent symbols; this scan terminates at the leftmost symbol of the candidate reducible substring.

At this point the relation '<' is required (STACKOK in parser flow chart) otherwise the stack is incorrect and production look-up and semantic calls are not performed and a diagnostic message is issued. The error recovery mechanism is activated by either an incorrect stack or the nonexistence of a production to match the candidate reducible substring.

The error recovery procedure first outputs the current contents of the stack. The stack is then examined from the leftmost symbol and compared to a recovery stack of maximum length 10 (this stack was processed by the syntax analyzer as SYM(1) --- SYM(10) and represents what is normally expected to reside in the stack). The symbols in the parsing stack (and their associated value) are kept as long as they match the recovery stack.

After the stack has been corrected, the input scanner is reset to the beginning of the current input file and the symbols are read and checked to see if they may occur adjacent; if they may not occur adjacent, a diagnostic message is issued

giving the symbols and how they were classified (WORD, STRING, INTEGER or RESERVED). This scanning continues until the SCAN_STOP symbol is detected. The symbols thus processed are erased from the input file and normal parsing is resumed.

Although this method is simple, it has proven quite useful for the types of languages implemented to date. It provides automatic diagnostics and recovery related to the input grammar with little effort of the user.

4. SEMANTIC CONSTRUCTOR

The semantic constructor processes its input text, which is a mixture of keywords and PL/1 statements, and generates a program which is compatible with the parser. Its purpose is to provide the standard procedure and parameter declarations and to construct the branching logic for selecting that portion of the code applicable for a particular production; the overall branching structure cannot be affected by the code for any production. The specification of the semantic constructor is given in Appendix D.

The syntax for the semantic constructor follows:

SEMANTICS : := *SEMANTICS* PROG-NAME CODA PRODUCTIONS

PROG-NAME : := procedure name to be given to these semantics

CODA : := *CODE* <block of PL/1 code> *END*

PRODUCTION : := INTERPRETATION

: := PRODUCTION INTERPRETATION

INTERPRETATION : := *PRODUCTION* INTEGER CODA

As the syntax illustrates, the basic unit is an INTERPRETATION, which is the keyword *PRODUCTION* followed by an integer followed by the keyword *CODE* followed by a block of PL/1 code terminated by *END*. For this unit, an if test on the integer is constructed and a label ("L" followed by the integer) attached to form a DO group for the block of PL/1 code. The end of the PL/1 block causes an END label to be generated, thereby closing the DO group.

The semantic constructor is implemented using the syntax analyzer and a skeleton parser with a hand coded semantic section. It will be used to illustrate the use of the SIMPLE system in Section 6.

5. POSSIBLE EXTENSIONS

5.1 Automatic Syntax Correction

Some grammars require the insertion of several artificial productions and renaming of variables in different parts of the grammar to be a simple precedence grammar. This results in the grammar's being longer and not in a form easily useable by users of a special language.

The methods of removing precedence violations discussed in Section 3.2 were developed with the idea of possible inclusion into the syntax analyzer; in fact, the organization of the syntax analyzer was modified to permit this insertion in an easy straightforward manner. Removing the conflicts automatically would make the grammars shorter and more readily useable. I see no problem in doing this, but haven't had the time to do so.

5.2 Parser Modification to Allow Simple Manipulation of the Parsing Stack by the Semantic Procedure

As discussed earlier (Section 3.5) the parser and semantics are separate and the semantics may operate only upon the value stack and not the associated parsing stack. This means that the system is entirely syntax driven and the parsing cannot be affected by any semantic meaning. Situations do arise where the parsing must be affected by the semantic meaning.

Consider for example the evaluation of an algebraic expression where the variables may stand for a numeric value or for some other algebraic expression. The parser only recognizes symbols and cannot determine whether a symbol represents a primitive or an intermediate expression; only the semantics can determine this. At this point the semantics need to defer a reduction and alter the stack (i.e., the semantics would like to replace the variable by its equivalent expression). This particular problem originated in the Graphic Description Language of GEMS (George 1969a, b).

The problem was to allow a form of stack manipulation which would still preserve a reasonable level of protection. From my work with and modification of the parser, I knew that all the error recovery and error diagnostics are based upon the symbol recognition; thus, the manipulation should be upon the input string so that the symbol recognizer can process the input string and thus preserve error recovery and diagnostics; this would provide the "reasonable level of protection."

The solution is to provide an external switch and string to both the parser and the semantics. When the semantics wants to erase the effect of a whole production and insert a string into the current input string (i.e., this new string is to be processed before the rest of the old string), the semantics leaves the string in the external string variable and sets the switch. Upon return from the semantics, the parser checks the switch and performs the ordinary reduction if the switch has not been set. If the switch has been set, the parser inserts the external string into the proper place of the current input string, resets the switch and erases the current production from the stack; it performs no reduction but resumes the normal parsing.

This solution not only provides a substitution facility for intermediate or non-basic primitives, but also allows grammars to be used with apparently disjunct productions. These disjunct productions can represent shortened or alternate forms of a production; these sometimes cause precedence violations and cannot be resolved in any other manner. For example in SPIRES (George 1967b; Parker 1967) it is desired to have

AUTHOR name AND name

to be equivalent to

AUTHOR name AND AUTHOR name.

This disjunct production method can be used for search classes other than AUTHOR by remembering the last search type and performing a substitution.

6. EXAMPLE APPLICATIONS OF SIMPLE

6.1 Semantic Constructor

For an example consider the semantic constructor. The syntax in simple precedence form and the data for SIMPLE's executive follow:

```
//GO.SYNDATA
    SEMANT_NAME='SEMANT'
/*
//GO.SYNTAX DD*
*SYNTAX*
SEMANTICS          *:==*    SEMANT CODA PRODUCTIONS *;*
PRODUCTIONS        *:==*    INTERPRETATIONS *NO-SEMANT*  *;*
SEMANT             *:==*    *SEMANTICS* WORD   *;*
INTERPRETATIONS    *:==*    INTERPRETATION *NO-SEMANT*  *;*
                   *:==*    INTERPRETATIONS INTERPRETATION
                                *NO-SEMANT*  *;*
INTERPRETATION     *:==*    INTERP *CODE*  *;*
INTERP             *:==*    *PRODUCTION*  INTEGER  *;*
CODA               *:==*    *CODE*
*END-SYNTAX*
/*
```

The semantics are:

```
//GO.SEMANTIC   DD*
    *SEMANTICS*  SEMANT  *CODE* *END*
    *PRODUCTION*  1  *CODE*
        PUT FILE (OUT)  EDIT  ('END' || VS(J)| ';')
            (Col  (10),  A);
        CLOSE FILE (OUT);
        *END*
    *PRODUCTION*  3  *CODE*
        PUT FILE (OUT)  EDIT
            (VS (K) || ':  PROC (N, VS, J, K, ANS, ERROR);')
            (Col (2),  A);
        PUT FILE (OUT) EDIT
            ('DCL (N, J, K, ANS) FIXED BIN, ',
            'VS(0:50) CHAR(400) VAR, ',
            'ERROR BIT(1);')
            (Col (10),  A,  Col (20),  A,  Col (20),  A);
        VS(J) = VS(K);
        *END*
    *PRODUCTION*  6  *CODE*
        PUT FILE (OUT) EDIT
            ('RETURN;',  'END' || 'L' |  VS(J) || ';')
            (2 (Col (10),  A));
        *END*
    *PRODUCTION*  7  *CODE*
        PUT FILE (OUT) EDIT
            ('IF N=',  VS(K),  ' THEN',  'L' || VS(K) || ':',
```

```
                'DO'; /*PRODUCTION NUMBER ',
                VS(K), '*/')
                (Col (10), 3 A, Col (2), A, Col (20), 3 A);
            VS(J) = VS(K);
            *END*
*END-SEMANTICS*
/*
```

An example input to this language is:

```
//GO.SOURCE
*SEMANTICS* SEM *CODE*
/* ANY PL/1 CODE CAN BE HERE*/
    *END*
*PRODUCTION* 1 *CODE*
    PUT FILE (OUT) EDIT
        ('PUT LIST (N) SKIP;')
        (Col (10), A);
    *END*
*PRODUCTION* 2 *CODE*
    PUT FILE (OUT) EDIT
        ('PUT LIST (N, J) SKIP;')
        (Col (10), A);
    *END*
*END-SEMANTICS*
/*
```

And the output is:

```
SEM:  PROC (N, VS, J, K, ANS, ERROR);
          DCL (N, J, K, ANS)  FIXED BIN,
                        VS(0:50) CHAR(400)VAR,
                        ERROR BIT (1);
          /* ANY PL/1 CODE CAN BE HERE */
          If N=1 THEN
L1:                 DO;  /*PRODUCTION NUMBER 1*/
              PUT LIST(N) SKIP;
          RETURN;
          END L1;
          IF N=2 THEN
L2:                 DO;  /*PRODUCTION NUMBER 2*/
              PUT LIST (N, J) SKIP;
          RETURN;
          END L2;
          END SEM;
```

6.2 A Command Language Meta System

During the Spring Quarter of 1970, a computer laboratory (CS 293) was organized by Professor W. F. Miller to allow small groups of students to participate in projects involving substantial programming tasks. Dr. Harry J. Saal and I led a group to study and implement a text editor system; the students were Howard Cohen, David Wyeth and Marice Schlumberger.

During the initial process of reviewing existing text editors, we arrived at the following conclusions:

1. No existing text editor had all the features desired;
2. We could not agree on a universal text editor language;
3. There was no existing system in which we could experiment with different text editor languages in an economical manner;
4. Generally, only one text editor was available in a computer system.

At this point, we realized that we were really talking about command languages rather than just text editor languages. The sentences of these languages are a command and consist of a command keyword followed by a list of parameters. Thus, we decided to design a meta system for defining command languages of this type.

The characteristics desired were:

1. The defined command language should be easy to change;
2. The system should be able to service various command languages.

The meta system developed for describing, scanning and implementing command languages (George and Saal 1971) has been used to define and implement two text editors (Schlumberger and Wyeth 1971) and will now be presented in detail.

6.2.1 The Model

The meta system consists of a table generator and a scanner. A specific command language is defined by a command description and the inclusion of any additional subroutines into the primitive library; the command description is

translated by the table generator to a form useable by the scanner as illustrated in Figure 7. The tables describe how a standard parameter list is to be constructed, thus allowing the primitive library members to be shared by various applications. The table generator provides a construction aid to a user with error diagnostics and some consistency checking.

To use a specific command language, the user designates to the scanner which table is to be used; this table is then obtained and saved in the user's work area. Commands can now be syntactically analyzed by the scanner using the specified table and the semantics of a command can be performed through activation of the appropriate subroutine in the primitive library. This is illustrated in Figure 8.

This model provides the versatility desired and allows command languages to be developed or modified modularly. New or modified commands can be tested without the other users of that particular command language system being aware of or affected by this testing. Further, each command language can be tailored to a user or group of users. This tailoring could provide simplified commands for less sophisticated users or could limit their actions or capabilities in items such as, read only systems, file access restrictions, etc.

6.2.2 The Table Generator

The Table Generator is implemented using SIMPLE and its definition is given in Appendix E. As indicated in the appendix, a command table consists of a set of options followed by a list of commands.

The options consist of the table name to which the table generator adds the current date and time for identification (this line is usually typed out when a user selects a table and, thus, indicates the version of the command system to the user), a separator to mark fields in the table (*PERIOD*) and a character which will inclose strings to indicate type <STRING>, (*QUOTES*).

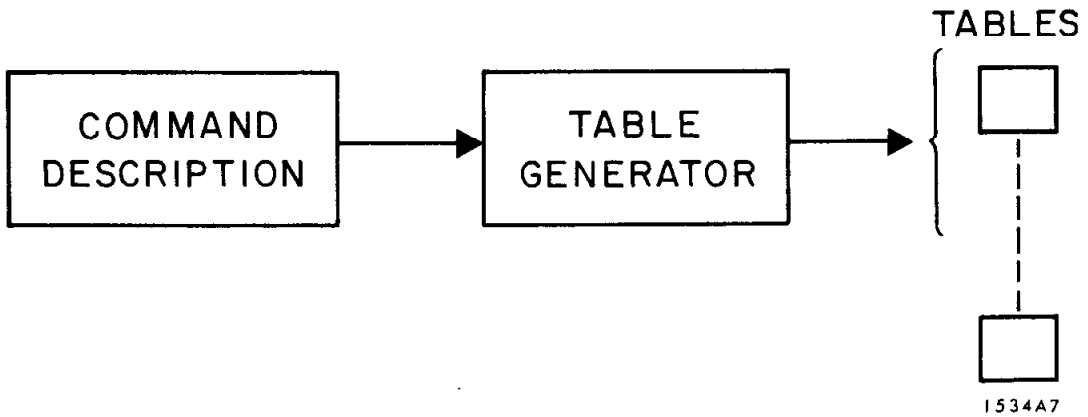


FIG. 7--Command language meta system - table generation.

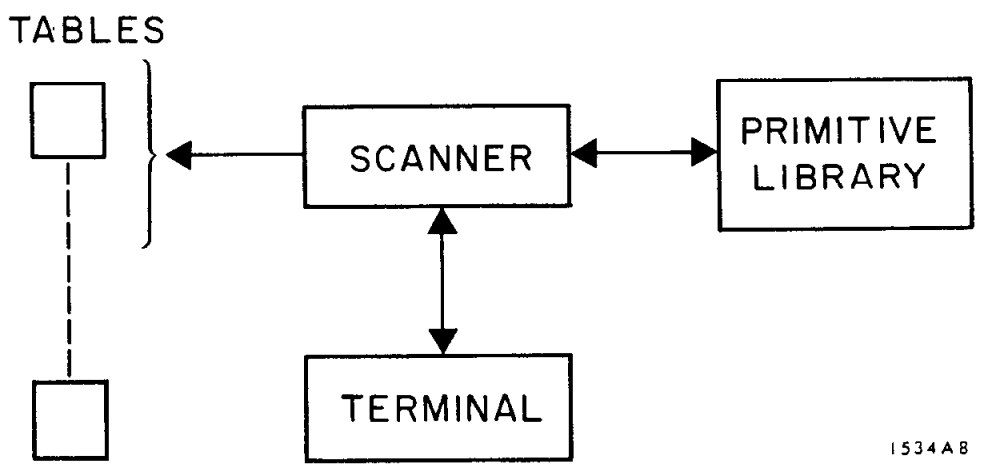


FIG. 8--Command language meta system - scanner.

The list of commands is composed of subroutines used by the commands and the commands, all are recursive. Commands are indicated by an identifier list followed by a parameter list; an identifier list is a list of identifier specifications; e.g.

KEYWORD LIST *RTN* SUB1 *DL-EX-LIST* "/" *DL-SKIP* "."

specifies a command whose name is LIST and whose semantic routine is named SUB1.

Normally, all special characters are treated as delimiters by the scanner; when scanning for the next item, the scanning proceeds until a delimiter is found and then the delimiter is deleted. In the above example, "/" is not to be deleted, but is to be returned as the following item; "." is not to be treated as a delimiter. Thus, 2/3 would be scanned as three items 2, / and 3 whereas 2:3 would be scanned as two, 2 and 3. Further, 2.3 would be scanned as one item 2.3.

Each parameter may be one of the following types

NUM	type number
STRING	type string
NAME	first letter alphabetic followed by alphanumerics
<STRING>	Call the table subroutine specified by <STRING>to obtain the parameter †

† The table subroutine calling mechanism is assumed to work by concatenating this <STRING> to the current unscanned input string and then activation of the scanner. This results in not only the subroutine activation, but character strings can be appended to the string. For example, if the current input pointer is at

	ABC) ↑
and the subroutine call is	"SUB5 ("
then, the input becomes	SUB5 (ABC) ↑

Further, parameters may be restricted by the options:

P	No parameter before the one with this option can be filled in after this parameter
K	This parameter can only be filled in after recognition of its key

and parameters may be initialized. A parameter may have multiple keys of the types:

VALUE	Take the next item after the key in the current input string and assign it to the parameter if it is of the proper type
VALUE ⟨STRING⟩	Take everything up to the occurrence of ⟨STRING⟩, assign it to the parameter and then delete ⟨STRING⟩ from the input
VALUESHORT ⟨STRING⟩	Take everything up to the occurrence of ⟨STRING⟩ and assign it to the parameter; do not delete ⟨STRING⟩ from the input
SELF ⟨STRING⟩	Assign ⟨STRING⟩ to the parameter
CALL ⟨STRING⟩	Call the table routine named in ⟨STRING⟩; same functioning as the previous subroutine call

For example, if the desired command is:

LIST
L ⟨NUM⟩ {/ ⟨NUM⟩} IN ⟨FILENAME⟩

where,

[. . .] means one of the options must be used; and

{. . .} means the contents are optional

The command description is:

```
*QUOTES*          *=*  "
*PERIOD*          *=*  .
*TBL-NAME*        *=*  "EXAMPLE"
*KEYWORD*  LIST  *RTN*  SUB1  *DL-EX-LIST*  "/"
*KEYWORD*  L     *RTN*  SUB1  *DL-EX-LIST*  "/"
    *PARM*   *NUM*   *INITIAL*  "-1"   *END*
    *PARM*   *NUM*   *K*   *P*   *INITIAL*   "-1"
          *KEY*   /   *VALUE*   *END*
    *PARM*   *NAME*   *K*   *P*   *INITIAL*  ""
          *KEY*   IN   *VALUE*   *END*
*END-TABLE*
```

6.2.3 The Scanner

The original scanner was designed to test the model and the design of the tables produced by the table generator (George and Saal 1971; the table generator is the author's work, the scanner work was done by H. J. Saal and the command description language and the tables were a joint effort). This scanner was then modified to perform the subroutine linkages to complete the meta system model as discussed in Section 6.2.1 (Schlumberger and Wyeth 1971). The original version of the scanner accepts an input string from the user and builds a parenthesized expression indication which subroutine is to be activated, number of characters scanned and a parameter list; if an error occurs, a diagnostic is given with a pointer to the offending character. This original version does provide a convenient testing vehicle for checking out the syntax of a command language and will be used for illustration.

6.2.4 Examples Using the Command Language Meta System

The system has been used to define and implement two text editors (Schlumberger and Wyeth 1971) and found to be an efficient way to experiment with different text editor languages. In particular, the syntax is easily debugged

and commands may be modified or added easily. Some example commands from each of these languages will be used for illustration.

6.2.4.1 WYLBUR Example

WYLBUR (---WYLBUR 1969) is a locally available text editor and several commands from it will be used as an example. The commands are:

1. List Command

$$\left\{\begin{matrix}\text{LIST}\\ \text{L}\end{matrix}\right\} \quad [\ \langle\text{ARANGE}\rangle\ ]\ [\ \text{IN}\ ]\ [\ \langle\text{NRANGE}\rangle\ ]$$

2. Change Command

$$\left\{\begin{matrix}\text{CHANGE}\\ \text{CH}\end{matrix}\right\}\ [\ \langle\text{ARANGE}\rangle\ ]\ \text{TO}\ [\ \langle\text{STRING}\rangle]\ [\ \text{IN}\ ]\ [\ \langle\text{NRANGE}\rangle\ ]$$

3. Copy Command

$$\left\{\begin{matrix}\text{COPY}\\ \text{CO}\end{matrix}\right\} \quad [\ \langle\text{NRANGE}\rangle\ ]\ \text{TO}\ [\ \langle\text{VALUE}\rangle\ ] \quad \text{BY}\ [\ \langle\text{NUMBER}\rangle\ ]$$

4. Set Command

SET [LENGTH =⟨NUMBER⟩] [DELTA = ⟨NUMBER⟩]
[UPLOW | UPPER | VERBOSE | TERSE]

where,

$$\langle\text{ARANGE}\rangle = \left\{\begin{matrix}\neg\ \langle\text{STRING}\rangle\\ \langle\text{STRING}\rangle\end{matrix}\right\}[\langle\text{NUMBER}\rangle[\ /\ \langle\text{NUMBER}\rangle]\ ]\ [(\langle\text{NUMBER}\rangle)]$$

$$\langle\text{NRANGE} = \langle\text{VALUE}\rangle \mid \langle\text{VALUE}\rangle\ /\ \langle\text{VALUE}\rangle \mid \langle\text{NRANGE}\rangle\ ,\ \langle\text{VALUE}\rangle \mid \langle\text{NRANGE}\rangle\ ,\ \langle\text{VALUE}\rangle\ /\ \langle\text{VALUE}\rangle$$

$$\langle\text{VALUE}\rangle = \langle\text{NUMBER}\rangle \mid \text{FIRST} \mid \text{LAST} \mid \text{END} \mid \text{ALL}$$

$$\langle\text{STRING}\rangle = \ '\ \langle\text{CHARACTER STRING}\rangle\ ' \mid\ ''\ \langle\text{CHARACTER STRING}\rangle\ ''$$

[..] means optional

{..} means one of the options must be present

The specification of the syntax of these commands is given in Appendix F with the resultant generated table. An example conversation with the scanner using the tables follows:

```
UNIT#?13
 WYLBUR EXAMPLE---GEORGE  07/17/70  14:33:48.260
LIST TABLES?no
LIST COMMANDS?yes
LIST
L
CHANGE
CH
COPY
CO
SET
COMMAND?list
(SUB1,5,(,),(0,0),)
COMMAND?l 1,2
(SUB1,6,(,((1),,((2),,))),(0,1),)
COMMAND?l 1/4
(SUB1,6,(,((1),(4),)),(0,1),)
COMMAND?l all
(SUB1,6,(,((-4),,)),(0,1),)
COMMAND?l 'y'
(SUB1,6,(((,(Y,-1,-1,-1))),),(1,0),)
COMMAND?l 'y' 1/8 (9) in all
(SUB1,21,(((,(Y,1,8,9))),((-4),,)),(1,1),)
COMMAND?list everything
*ERROR*       |
COMMAND?set terse
(SUB4,10,(,,4),(0,0,1),)
COMMAND?set delta=12
(SUB4,13,((12),,0),(1,0,0),)
COMMAND?set delta=1 length=2 terse
(SUB4,27,((1),(2),4),(1,1,1),)
COMMAND?change 'sk' to 'wk' in all
(SUB2,27,(((,(SK,-1,-1,-1))),(WK),((-4),,)),(1,1,1),)
COMMAND?ch ¬'t' 4/9   (8) to "e" in all
(SUB2,32,(((¬,(T,4,9,8))),(E),((-4),,)),(1,1,1),)
COMMAND?copy 1/5 to 16.2
(SUB3,17,(((1),(5),),(16.2),-1),(1,1,0),)
COMMAND?*RESTART*
```

6.2.4.2 CRBE Example

CRBE (Wells 1970a and b) is another locally available text editor and several commands from it will be illustrated. The commands are:

1. List Command

$$\left\{\begin{matrix}\text{LIST}\\ \text{L}\end{matrix}\right\} \quad [\ \langle\text{NRANGE}\rangle\ ]\ \ [\ \langle\text{ARANGE}\rangle\ ]$$

2. Save Command

$$\left\{\begin{matrix}\text{SAVE}\\ \text{S}\end{matrix}\right\} \quad [\ \langle\text{FNAME}\rangle\ ]\ \ [\ (\ \langle\text{NUMBER}\rangle\ [\ ,\ \langle\text{NUMBER}\rangle\ ]\ )\ ]$$
$$[\ \text{KEEP}\ |\ \text{PURGE}\ ]\ [\ \text{REPLACE}\ |\ \text{REPL}\ ]$$

3. Bring Command

$$\left\{\begin{matrix}\text{BRING}\\ \text{B}\end{matrix}\right\} \quad \left\{\begin{matrix}\langle\text{NUMBER}\rangle\\ \langle\text{NAME}\rangle\\ [\text{D}\ |\ \text{DSNAME}] = \langle\text{FNAME}\rangle\ [\ (\langle\text{NAME}\rangle)\ ]\ [\ ,\ [\text{V}\ |\ \text{VOL}] = \langle\text{NAME}\rangle\ ]\end{matrix}\right\}$$
$$[\ \text{SEQ}\ |\ \text{NOSEQ}\ ]$$

4. Change Command

$$\left\{\begin{matrix}\text{CHANGE}\\ \text{CH}\end{matrix}\right\}\ [\ \langle\text{NRANGE}\rangle\ ]\ [\ \neg\langle\text{STRING}\rangle\ |\ \langle\text{STRING}\rangle\ ]\ [\ \langle\text{STRING}\rangle\ ]$$
$$[\ \text{COL} = (\ \langle\text{NUMBER}\rangle\ [\ ,\ \langle\text{NUMBER}\rangle\ ]\ )\ ]$$
$$[\ \text{NOTEXT}\ |\ \text{NOLIST}\ ]$$

where,

$$\langle\text{NRANGE}\rangle = [\ \langle\text{NUMBER}\rangle\ |\ \text{FIRST}\ ]\ [\ \langle\text{NUMBER}\rangle\ |\ \text{LAST}\ ]\ [(\ \langle\text{NUMBER}\rangle\ )]$$

$$\langle\text{ARANGE}\rangle = \left\{\begin{matrix}\neg\langle\text{STRING}\rangle\\ \langle\text{STRING}\rangle\end{matrix}\right\} [\text{COL} = (\ \langle\text{NUMBER}\rangle\ [\ ,\ \langle\text{NUMBER}\rangle\ ]\ )\ ]$$
$$[\ \text{SEQ}\ |\ \text{NOSEQ}\ ]$$

$$\langle\text{FNAME}\rangle = \langle\text{NAME}\rangle\ |\ \langle\text{FNAME}\rangle\ .\ \langle\text{NAME}\rangle$$

$\langle\text{NAME}\rangle$ = First character alpha rest alpha-numeric

$\langle\text{STRING}\rangle$ = ' $\langle$CHARACTER STRING$\rangle$ ' | " $\langle$CHARACTER STRING$\rangle$ "

[...] means optional

{...} means one of the options is required

The specification of the syntax of these commands is given in Appendix G with the resultant generated table. An example conversation with the scanner using the tables follows:

```
UNIT#?15
 CRBE EXAMPLE---GEORGE  07/22/70  12:50:25.960
LIST TABLES?no
LIST COMMANDS?yes
LIST
L
SAVE
S
BRING
B
CHANGE
CH
COMMAND?list
(SUB1,5,(,),(0,0),)
COMMAND?list 1/4
(SUB1,9,(((1),(4),-1),),(1,0),)
COMMAND?l 1,4
(SUB1,6,(((1),(4),-1),),(1,0),)
COMMAND?l 'y' in all
*ERROR*       |
COMMAND?l 'y'
(SUB1,6,(,((,(Y,,0)))),(0,1),)
COMMAND?l first last
(SUB1,13,(((0),(-2),-1),),(1,0),)
COMMAND?l 1,2,(9),¬"k",col=(2,3)
(SUB1,25,(((1),(2),(9)),((¬,(K,((2,3)),0)))),(1,1),)
COMMAND?l 1 2 (9) ¬"K" col=(2,3)
(SUB1,25,(((1),(2),(9)),((¬,(K,((2,3)),0)))),(1,1),)
COMMAND?save ,repl
(SUB2,11,((ACTIVE,),,-1,0),(1,0,0,1),)
COMMAND?save ss.dd.ff,v=wyl003,(200,500)
*ERROR*                 |
COMMAND?save ss.dd.ff,vol=wyl003,(200,500)
*ERROR*                 |
COMMAND?save ss.dd.ff,repl
(SUB2,19,((SS,(DD,(FF,))),,-1,0),(1,0,0,1),)
COMMAND?b 123
(SUB3,6,(,-1,,123),(0,0,0,1),)
COMMAND?b jegxx123
(SUB3,11,(,-1,JEGXX123,),(0,0,1,0),)
COMMAND?b d=ss.dd(member),v=wyl003
(SUB3,27,(((SS,(DD,)),MEMBER,((WYL003))),-1,,),(1,0,0,0),)
COMMAND?ch 1,2,'y','u',nolist
(SUB4,22,(((1),(2),-1),(,(Y)),(U),,1),(1,1,1,0,1),)
COMMAND?*quit*
GOODBYE!
?
```

BIBLIOGRAPHY

1. Feldman, Jerome A. and Gries, David (1967). Translator Writing Systems. Computer Science Department, Stanford University, Technical Report No. CS 69. Also appeared in Comm. ACM, 11(1968), 2(February), 77-113.

2. Fischer, Michael J. (1969). Some Properties of Precedence Languages. ACM Symposium on Theory of Computing, 181-190.

3. George, J.E. (1967a). SARPSIS: Syntax Analyzer, Recognizer, Parser and Semantic Interpretation System. Stanford Linear Accelerator Center, CGTM 34, November 15, 1967.

4. George, J.E. (1967b). The SPIRES Scope Demonstration System. Stanford Linear Accelerator Center, CGTM 33, November 15, 1967.

5. George, James E. (1969a). The System Specification of GLAF: A Linear String Graphical Language Facility. Stanford Linear Accelerator Center, GSG-61, February, 1969.

6. George, J.E. (1969b). GEMS: A Graphic Experimental Meta-System. Stanford Linear Accelerator Center, GSG 63, June, 1969.

7. George, J.E. (1969c). Rules for Transforming a Grammar to a Simple Precedence Grammar Utilizing Artificial Productions. Stanford Linear Accelerator Center Computation Group, GSG-62, July, 1969.

8. George, James E. and Saal, Harry J. (1971). A Command Language Meta-System. Fourth Hawaii International Conference on System Sciences , 483-485. Also Stanford Linear Accelerator Center, SLAC-PUB-844.

9. Gray, James (1969). Precedence Parsers for Programming Languages. Department of Computer Science, University of California.

10. Learner, A. and Lim, A.L. (1970). A Note on Transforming Context-Free Grammars to Wirth-Weber Precedence Form. The Computer Journal, 13, 2(May), 142-144.

11. Leinius, Ronald Paul (1970). Error Detection and Recovery for Syntax Directed Compiler Systems. University of Wisconsin.

12. McAfee, J. and Presser, L. (1970). An Algorithm for the Design of Simple Precedence Grammars. Department of Electrical Engineering, University of California at Santa Barbara.

13. Parker, Edwin B. (1967). SPIRES 1967 Annual Report. Institute for Communication Research, Stanford University, December, 1967.

14. Presser, L. (1968). The Structure, Specifications and Evaluation of Translators and Translator Writing Systems. Department of Engineering, University of California at Los Angeles, Report No. 68-51.

15. Presser, L. and Melkanoff M.A. (1969). Transformation to Simple-Precedence. Second Hawaii International Conference on System Science, 695-698.

16. Schlumberger, Maurice and Wyeth, David (1971). A Multi-Editor System. Computer Science Department, Stanford University, CS 293 Report. Also, Stanford Linear Accelerator Center, CGTM 127.

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

18. Wells, J. (1970a). CRBE Command List. SLAC Facility, Stanford Computation Center, Stanford University, User Note 39.

19. Wells, J. (1970b). CRBE Commands. SLAC Facility, Stanford Computation Center, Stanford University.

20. Wirth, Niklaus and Weber, Helmut (1966a). EULER: A Generalization of ALGOL, and its Formal Definition: Part I. Comm. ACM, 9, 1(January), 13-25.

21. Wirth, Niklaus and Weber, Helmut (1966b). EULER: A Generalization of ALGOL, and its Formal Definition: Part II. Comm. ACM, 9, 2(February), 89-99.

22. ------ (1969). WYLBUR Reference Manual. Campus Facility, Stanford Computation Center, Stanford University, Appendix E, User's Manual.

APPENDIX A

SIMPLE'S EXECTIVE

```
SIMPLE: PROC OPTIONS (MAIN);
        DCL FILE1 CHAR(8) VAR, /*SYNTAX EQUATIONS INPUT FILE*/
            FILE2 CHAR(8) VAR, /*PARSING PROGRAM INPUT FILE*/
            FILE3 CHAR(8) VAR, /*PARSING PROGRAM OUTPUT FILE*/
            FILE4 CHAR(8) VAR, /*SYNTAX OUTPUT FILE*/
            FILE5 CHAR(8) VAR, /*SYNTAX DATA OPTIONS*/
            FILE6 CHAR(8) VAR, /*SEMANTIC INPUT FILE*/
            FILE7 CHAR(8) VAR, /*SEMANTIC DIAGNOSTIC OUTPUT FILE*/
            FILE8 CHAR(8) VAR, /*SEMANTIC PROGRAM OUTPUT FILE*/
            SINIT CHAR(20) VAR, /*INITIATOR FOR SYNTAX ANALYZER*/
            SSEP CHAR(20) VAR, /*SEPARATOR FOR LEFT-RIGHT SIDES*/
            STERM CHAR(20) VAR, /*TERMINATOR FOR EQUATIONS*/
            SEND CHAR(20) VAR, /*TERMINATOR FOR SYNTAX*/
            SSEMANT CHAR(20) VAR, /*INDICATES NO SEMANTICS FOR THIS
                                    PRODUCTION*/
            PARSER_NAME CHAR(8), /* NAME TO BE SUBSTITUTED FOR
                  *PARSER* IN FILE2 */
            SEMANT_NAME CHAR(8), /* NAME TO BE SUBSTITUTED FOR
                  *SEMANT* IN FILE2 */
            INTEGER CHAR(20) VAR,/*THAT SYMBOL USED IN SYNTAX FOR
                                    AN INTEGER*/
            WORD CHAR(20) VAR, /*THAT SYMBOL USED IN SYNTAX FOR WORD*/
            STRING CHAR(20) VAR, /*THAT SYMBOL USED FOR STRING */
            QUOTES CHAR(20) VAR, /*THAT SYMBOL USED FOR QUOTES*/
            SEQUENCE CHAR(20) VAR, /*THE INITIAL SYMBOL OF THE SYNTAX
                        WHEN IT OCCURRS IN THE STACK THE PARSING
                        IS TERMINATED    */
            TERMINAL CHAR(20) VAR, /*THAT SYMBOL USED TO FORCE PARSING
                                    TO BE COMPLETED    */
            ERRORSCAN CHAR(20) VAR, /*THAT SYMBOL IN THE SYNTAX WHICH
                         IS USED IN ERROR RECOVERY..THE TEXT BETWEEN
                         TWO OF THESE SYMBOLS IS EFFECTIVELY DELETED*/
            SYM(10) CHAR(20) VAR, /*THOSE SYMBOLS WHICH ARE EXPECTED
                      TO RESIDE IN THE I-TH POSITION OF THE PARSING
                      STACK  */
            SCAN_STOP CHAR(20) VAR, /*THAT SYMBOL IN THE SYNTAX WHICH
                              UPON ENTRY INTO THE PARSING STACK CAUSES
                              ALL INPUT TO BE IGNORED BY THE PARSER
                              UNTIL THE SYMBOL AFTER SCAN_START.*/
            SCAN_START CHAR(20) VAR, /* RESTARTS THE PARSING AFTER THE
                               APPEARANCE OF THIS SYMBOL*/
            MLIM FIXED BIN, /*MAXIMUM NUMBER OF SYMBOLS*/
            MMLIM FIXED BIN, /*MAXIMUM NUMBER OF NON-BASIC SYMBOLS*/
            NLIM FIXED BIN, /*MAXIMUM NUMBER OF PRODUCTIONS*/
            RLIM FIXED BIN; /*MAXIMUM NUMBER OF RIGHT ELEMENTS*/
        DCL I FIXED BIN;
        ON ENDFILE(SYNDATA) GO TO XXX;
        FILE1='SYNTAX';FILE2='SPARSER';FILE3='PARSER';FILE4='PSYNTAX';
        FILE5='SYNDATA'; FILE6='SEMANTICS'; FILE7='PSEMANT';
        FILE8='SEMANT'; PARSER_NAME='SEMANT'; SEMANT_NAME='CODE_OUT';
        SINIT='*SYNTAX*'; SSEP='*::=*'; STERM='*;*';
        SEND='*END-SYNTAX*'; SSEMANT='*NO-SEMANT*';
           INTEGER='INTEGER';  WORD='WORD'; QUOTES='"'; MLIM=20;
         MMLIM=20;  NLIM=20;     RLIM=8;    STRING='STRING';
        SEQUENCE='SEMANTICS'; TERMINAL='*END-SEMANTICS*';
        ERRORSCAN='*END*';
        DO I=1 TO 10;  SYM(I)='';  END;
            SCAN_START='*END*';   SCAN_STOP='*CODE*';
        SYM(1)='SEMANT'; SYM(3)='INTERPRETATIONS'; SYM(2)='CODA';
        OPEN FILE (SYNDATA) TITLE(FILE5) INPUT STREAM;
        GET FILE(SYNDATA) DATA;
XXX:    CALL SYNTAX (FILE1,FILE2,FILE3,FILE4,SINIT,SSEP,STERM,SEND,
                     SSEMANT,PARSER_NAME,SEMANT_NAME,
                INTEGER,WORD,STRING,QUOTES,SEQUENCE,TERMINAL,ERRORSCAN,
                     SYM,SCAN_STOP,SCAN_START,MLIM,MMLIM,NLIM,RLIM);
     CALL SEMANT(FILE6,FILE8,FILE7);
         END SIMPLE;
```

APPENDIX B

SYNTAX ANALYZER

```
SYNTAX:   PROC(FILE1,FILE2,FILE3,FILE4,SINIT,SSEP,STERM,SEND,SSEMANT,
              PARSER_NAME,SEMANT_NAME,
              INTEGER,WORD,STRING,QUOTES,SEQUENCE,TERMINAL,ERRORSCAN,
                   SYM,SCAN_STOP,SCAN_START,MLIM,MMLIM,NLIM,RLIM);
          DCL FILE1 CHAR(8) VAR, /*SYNTAX EQUATIONS INPUT FILE*/
              FILE2 CHAR(8) VAR, /*PARSING PROGRAM INPUT FILE*/
              FILE3 CHAR(8) VAR, /*PARSING PROGRAM OUTPUT FILE*/
              FILE4 CHAR(8) VAR, /*SYNTAX OUTPUT FILE*/
              SINIT CHAR(20) VAR, /*INITIATOR FOR SYNTAX ANALYZER*/
              SSEP CHAR(20) VAR, /*SEPARATOR FOR LEFT-RIGHT SIDES*/
              STERM CHAR(20) VAR, /*TERMINATOR FOR EQUATIONS*/
              SEND CHAR(20) VAR, /*TERMINATOR FOR SYNTAX*/
              SSEMANT CHAR(20) VAR, /*INDICATES NO SEMANTICS FOR THIS
                                      PRODUCTION*/
              PARSER_NAME CHAR(8),/*NAME TO BE SUBSTITUTED FOR
                    *PARSER* IN FILE2 */
              SEMANT_NAME CHAR(8), /*NAME TO BE SUBSTITUTED FOR
                    *SEMANT* IN FILE2 */
              INTEGER CHAR(20) VAR,/*THAT SYMBOL USED IN SYNTAX FOR
                                     AN INTEGER*/
              WORD CHAR(20) VAR, /*THAT SYMBOL USED IN SYNTAX FOR WORD*/
              STRING CHAR(20) VAR, /*THAT SYMBOL IN SYNTAX FOR STRING */
              QUOTES CHAR(20) VAR, /*THAT SYMBOL USED FOR QUOTES*/
              SEQUENCE CHAR(20) VAR, /*THE INITIAL SYMBOL OF THE SYNTAX
                           WHEN IT OCCURRS IN THE STACK THE PARSING
                           IS TERMINATED    */
              TERMINAL CHAR(20) VAR, /*THAT SYMBOL USED TO FORCE PARSING
                                       TO BE COMPLETED    */
              ERRORSCAN CHAR(20) VAR, /*THAT SYMBOL IN THE SYNTAX WHICH
                            IS USED IN ERROR RECOVERY..THE TEXT BETWEEN
                            TWO OF THESE SYMBOLS IS EFFECTIVELY DELETED*/
              SYM(10) CHAR(20) VAR, /*THOSE SYMBOLS WHICH ARE EXPECTED
                        TO RESIDE IN THE I-TH POSITION OF THE PARSING
                        STACK */
              SCAN_STOP CHAR(20) VAR, /*THAT SYMBOL IN THE SYNTAX WHICH
                                UPON ENTRY INTO THE PARSING STACH CAUSES
                                ALL INPUT TO BE IGNORED BY THE PARSER
                                UNTIL THE SYMBOL AFTER SCAN_START.*/
              SCAN_START CHAR(20) VAR, /* RESTARTS THE PARSING AFTER THE
                                 APPEARANCE OF THIS SYMBOL*/
              MLIM FIXED BIN, /*MAXIMUM NUMBER OF SYMBOLS*/
              MMLIM FIXED BIN, /*MAXIMUM NUMBER OF NON-BASIC SYMBOLS*/
              NLIM FIXED BIN, /*MAXIMUM NUMBER OF PRODUCTIONS*/
              RLIM FIXED BIN; /*MAXIMUM NUMBER OF RIGHT ELEMENTS*/
           DCL XINTEGER FIXED BIN, /*NUMBER FORM OF INTEGER*/
               XSCAN_STOP FIXED BIN, /*NUMBER FORM OF SCAN_STOP*/
               XSEQ FIXED BIN, /*NUMBER FORM OF SEQUENCE*/
               XSYM(10) FIXED BIN, /*NUMBER FORM OF SYM(*)  */
               XTERM FIXED BIN, /*NUMBER FORM OF TERMINAL*/
               XSTRING FIXED BIN, /*NUMBER FORM OF STRING */
               XWORD FIXED BIN; /*NUMBER FORM OF WORD*/
          DCL M FIXED BINARY;                 /* NUMBER OF SYMBOLS  */
          DCL MM FIXED BINARY;                /* NO NON-BASIC SYMBOLS */
          DCL N FIXED BINARY;                /* NUMBER OF PRODUCTIONS*/
          DCL SYT(0:MLIM) CHAR(20) VAR;/*SYMBOL TABLE                  */
```

```
        DCL PRD(NLIM,0:RLIM) FIXED BIN;  /* PRD IN NUMBER FORM*/
        DCL P(NLIM,0:RLIM) CHAR(20) VAR; /*PRODUCTIONS IN STRING FORM*/
        DCL SEMANT(NLIM) BIT(1); /*TRUE IF NO SEMANTICS FOR ITH PROD*/
        DCL H(0:MLIM,0:MLIM) CHAR(1); /*PRECEDENCE MATRIX     */
        DCL L(0:MMLIM,0:MLIM) BIT(1),R(0:MMLIM,0:MLIM) BIT(1);
                /*  L(I,J) TRUE MEANS THAT SY-J OCCURS IN THE */
                /* LEFT SYMBOL SET OF SY-I.R(I,J) MEANS THAT  */
                /*  SY-J IS IN RIGHT OF SY-I*/
        DCL (KEY(0:MLIM),PRTB(0:5*NLIM)) FIXED BIN;
        DCL BASVAL(MLIM) FIXED BIN;
        DCL BASSYM(MLIM) CHAR(20) VAR;
READ_SYNTAX_INPUT:          PROC;
        DCL INBUF CHAR(100) VAR,BUF CHAR(100) VAR,(I,K) FIXED BIN;
        /*READS SYNTAX INPUT AND MAKES UP P MATRIX AND SEMANT*/
DELETE: PROC(INBUF)  RETURNS(CHAR(100) VAR);
        /*DELETES LEADING BLANKS--RETURNS NULL IF ALL BLANK*/
        DCL (INBUF,STR) CHAR(100) VAR;
           STR=INBUF;
           IF STR='' | STR=' ' THEN RETURN('');
           DO WHILE (SUBSTR(STR,1,1)=' ');
               STR=SUBSTR(STR,2);
               END;
           RETURN(STR);
           END DELETE;
NEXT:   PROC  RETURNS(CHAR(100) VAR);
        DCL OUTA CHAR(100) VAR;
        /*FETCHES NEXT SYMBOL FROM INPUT*/
          ON ENDFILE(DATA) BEGIN;
             PUT FILE(OUT) EDIT('*****ENDFILE SYNTAX INPUT--NO ',
                 SEND)(SKIP,2 A);
             GO TO EXIT;
             END;
          IF INBUF='' THEN DO;
      LOOP:   GET FILE(DATA) EDIT (INBUF)(A(80));
              INBUF=INBUF||'  ';
              INBUF=DELETE(INBUF);
              IF INBUF='' THEN GO TO LOOP;
              END;
          I=INDEX(INBUF,' ');
          OUTA=SUBSTR(INBUF,1,I-1);
          INBUF=DELETE(SUBSTR(INBUF,I+1));
          RETURN(OUTA);
          END NEXT;
        DCL NEXT INTERNAL ENTRY RETURNS(CHAR(100) VAR),
         DELETE INTERNAL ENTRY(CHAR(100) VAR) RETURNS(CHAR(100) VAR);
           K=0;  N=1;  BUF='';     INBUF='';
           DO I=1 TO NLIM;P(I,0)='';SEMANT(I)='0'B; END;
           OPEN FILE(DATA) TITLE(FILE1) INPUT STREAM;
           DO WHILE (BUF ¬= SINIT);
              BUF=NEXT;
              END;
           BUF=NEXT;
           DO WHILE (BUF ¬= SEND);
               IF BUF=SSEP THEN K=1;
               ELSE IF BUF=STERM THEN DO;
                    DO I=K TO RLIM; P(N,I)='';    END;
                    IF N < NLIM THEN N=N+1;
                    K=0;
                    END;
               ELSE IF BUF=SSEMANT THEN SEMANT(N)='1'B;
```

```
                ELSE DO;
                     P(N,K)=BUF;
                     IF K < RLIM THEN K=K+1;
                     END;
                BUF=NEXT;
                END;
  EXIT:     CLOSE FILE(DATA);
            DO I=2 TO N;  IF P(I,0)='' THEN P(I,0)=P(I-1,0);  END;
            END READ_SYNTAX_INPUT;
BASIC:   PROC;
         /*MAKES SYMBOL TABLE AND NUMERICAL PRODUCTION TABLE PRD*/
         DCL (I,J,K) FIXED BIN;
            M=0;  SYT(0)='';
            DO K=0 TO RLIM;
                DO I=1 TO N;
                   DO J=0 TO M; IF P(I,K)=SYT(J) THEN GO TO FF; END;
                   M=M+1; J=M; SYT(M)=P(I,K);
      FF:          PRD(I,K)=J;
                   END;
                IF K=0 THEN MM=M;
                END;
            END BASIC;
COMP_KEY_PRTB:      PROC;
         /*COMPUTES KEY AND PRTB TABLES...KEY(I) REPRESENTS, FOR THE
           ITH SYMBOL THE INDEX INTO PRTB WHERE THOSE PRODUCTION ARE
           LISTED WHOSE RIGHT PART BEGINS WITH THE ITH SYMBOL..
           FOR EACH PRODUCTION, THE RIGHT PART IS LISTED WITHOUT
           ITS LEFTMOST SYMBOL, FOLLOWED BY THE NEGATIVE OF THE
           PRODUCTION NUMBER(IF NO SEMANTICS OPTION SELECTED N
           IS SUBTRACTED FROM THE PRODUCTION NUMBER) AND THE LEFT
           PART SYMBOL OF THE PRODUCTION.  ALL SYMBOLS ARE IN NUMERIC
           FORM.  THE END OF A LIST OF PRODUCTIONS REFERENCED BY KEY(I)
           IS MARKED WITH A 0 ENTRY IN PRTB.     */
           DCL (I,J,K,U,V) FIXED BIN;
             K=0; V=0;  KEY(0)=0;  PRTB(0)=0;
             DO I=1 TO M+1;
                 IF V¬=0 THEN KEY(I-1)=V;
                 V=0;
                 IF PRTB(K)¬=0 THEN K=K+1;
                 PRTB(K)=0;  KEY(I)=K;
                 DO J=1 TO N;
                     IF PRD(J,1)=I THEN DO;
                         IF V=0  THEN V=K+1;
                         DO U=2 TO RLIM;
                             IF PRD(J,U)¬=0 THEN DO;
                             K=K+1;  PRTB(K)=PRD(J,U);          END;
                             END;
                         K=K+1;
                         IF SEMANT(J) THEN PRTB(K)=-N-J;
                             ELSE PRTB(K)=-J;
                         K=K+1;       PRTB(K)=PRD(J,0);
                         END;
                     END;
                 END;
             END COMP_KEY_PRTB;
SYNTAX_OUTPUT:      PROC;
         /*OUTPUTS SYNTAX INFORMATION*/
         DCL (I,J,K) FIXED BIN;
         /*OUTPUT PRODUCTIONS IN STRING FORM*/
            PUT FILE(OUT) EDIT('PRODUCTIONS',' ')(PAGE,A,SKIP,A);
```

```
            DO I=1 TO N;
               PUT FILE(OUT) EDIT(I,P(I,0),SSEP)(SKIP,F(4),X(2),A,X(2),
                     A);
               DO J=1 TO RLIM;
                  IF P(I,J)¬='' THEN PUT FILE(OUT) EDIT(P(I,J))
                        (X(2),A);
                  END;
               IF SEMANT(I) THEN PUT FILE(OUT) EDIT ('*NO-SEMANTICS*')
                  (X(5),A);
               END;
            /*OUTPUT BASIC AND NON-BASIC SYMBOLS*/
               PUT FILE(OUT) EDIT('BASIC SYMBOLS','NON-BASIC SYMBOLS',' ')
                     (PAGE,A,COLUMN(50),A,SKIP,A);
               DO I=1 TO MAX(MM,M-MM);
                  IF I+MM<=M THEN PUT FILE(OUT) EDIT(MM+I,SYT(MM+I))
                        (SKIP,F(4),X(2),A);
                  IF I<=MM THEN PUT FILE(OUT) EDIT(I,SYT(I))
                        (COLUMN(50),F(4),X(2),A);
                  END;
            /*OUTPUT KEY AND PRTB*/
               PUT FILE(OUT) EDIT('I','KEY(I)','PRTB(KEY(I))',' ')
                     (PAGE,A,COLUMN(10),A,COLUMN(20),A,SKIP,A);
               DO I=1 TO M;
                  PUT FILE(OUT) EDIT(I,KEY(I),' ')(SKIP,F(4),COLUMN(10),
                        F(5),COLUMN(20),A);
                  DO K=KEY(I) BY 1 WHILE (PRTB(K)¬=0);
                     PUT FILE(OUT) EDIT(PRTB(K))(X(1),F(4));
                     END;
                  END;
             END SYNTAX_OUTPUT;
PRECEDENCE:  PROC;
            /*     FIND   H  PRECEDENCE MATRIX    */
            DCL (I,J,K) FIXED BIN,ERRORFLAG BIT(1);
            DCL (U,V,P,Q,A,B) FIXED BIN;
            DCL NN FIXED BIN, CHANGE BIT(1);
            DCL (C1(0:MLIM),C2(0:MLIM)) FIXED BIN;
                  /* THE IT'H SYMBOL OCCURS C1(I) TIMES AS LEFT  */
                  /*  AND C2(I) TIMES AS RIGHT                   */
            DCL (B1(0:NLIM),B2(0:NLIM)) BIT(1);
                  /*  B(K) MEANS THAT THE K'TH PRODUCTION HAS BEEN  */
                  /* ELIMINATED          */
            DCL (SO(0:NLIM),SL(0:NLIM),SR(0:NLIM)) FIXED BIN;
ENTER:      PROC (X,Y,S);
               DCL T CHAR(1);
               DCL (X,Y) FIXED BINARY,S CHAR(1);
               T=H(X,Y);
               IF T¬=' ' & T¬=S THEN DO;
                  IF ¬ ERRORFLAG THEN PUT FILE(OUT) EDIT
                     ('HINTS REGARDING PRECEDENCE VIOLATIONS',' ')
                     (PAGE,A,SKIP,A);
                  ERRORFLAG='1'B;
                  PUT FILE(OUT) EDIT
                        (U,SYT(X),T,S,SYT(Y))(SKIP,F(4),X(2),A,X(2),
                              2 A,X(2),A);
                  END;
               H(X,Y)=S;
               END ENTER;
            DO I=1 TO M;  C1(I)=0;  C2(I)=0;   END;
BA:         DO K=1 TO N;
               SO(K)=PRD(K,0);  SL(K)=PRD(K,1);  J=RLIM;
```

```
            DO WHILE (PRD(K,J)=0);  J=J-1;  END;
            SR(K)=PRD(K,J);  B1(K)='1'B;  B2(K)='1'B;
            C2(SO(K))=C1(SO(K))+1;    C1(SO(K))=C2(SO(K));
            END BA;
        DO I=1 TO MM;
            DO J=1 TO M;  R(I,J)='0'B;  L(I,J)='0'B;  END;    END;
        NN=N;   CHANGE='1'B;
BB:     DO WHILE (CHANGE & NN>0);
            CHANGE='0'B;
            DO K=1 TO N;
                IF B1(K) THEN DO;
                    A=SO(K);  B=SL(K);
                    IF ¬ L(A,B) THEN DO;  L(A,B)='1'B; CHANGE='1'B;
                        END;
                    IF B<=MM THEN DO J=1 TO M;
                        IF ¬ L(A,J) THEN IF L(B,J) THEN DO;
                            L(A,J)='1'B;  CHANGE='1'B; END; END;
                IF C1(B)=0 THEN DO; B1(K)='0'B; C1(A)=C1(A)-1;
                    NN=NN-1;    END;
            END BB;
        NN=N;   CHANGE='1'B;
BC:     DO WHILE (CHANGE & NN>0);
            CHANGE='0'B;
            DO K=1 TO N;
                IF B2(K) THEN DO;
                    A=SO(K);  B=SR(K);
                    IF ¬ R(A,B) THEN DO; R(A,B)='1'B; CHANGE='1'B;END;
                    IF B<=MM THEN DO J=1 TO M;
                        IF ¬ R(A,J) THEN IF R(B,J) THEN DO;
                            R(A,J)='1'B;  CHANGE='1'B;  END; END;
                    IF C2(B)=0 THEN DO;
                        B2(K)='0'B; C2(A)=C2(A)-1; NN=NN-1;  END;
            END BC;
        /*OUTPUT RIGHT AND LEFT SYMBOL SETS*/
           PUT FILE(OUT) EDIT ('RIGHT SYMBOL SETS',' ')(PAGE,A,SKIP,A);
           DO I=1 TO MM;
              PUT FILE(OUT) EDIT (SYT(I),'=')(SKIP,A,X(1),A);
              DO J=1 TO M;
                 IF R(I,J) THEN PUT FILE(OUT) EDIT (SYT(J))(X(2),A);
                 END;
              END;
           PUT FILE(OUT) EDIT('LEFT SYMBOL SETS',' ')(SKIP(4),A,SKIP,
                 A);
           DO I=1 TO MM;
              PUT FILE(OUT) EDIT (SYT(I),'=')(SKIP,A,X(1),A);
              DO J=1 TO M;
                 IF L(I,J) THEN PUT FILE(OUT) EDIT(SYT(J))(X(2),A);
              END;
           END;
        /*     FIND   H  PRECEDENCE MATRIX     */
        DO I=0 TO M; DO J=0 TO M; H(I,J)=' ';END; END;
        ERRORFLAG='0'B;
UV:     DO U=1 TO N;
            DO V=2 TO RLIM;
                IF PRD(U,V)¬=0 THEN DO;
                    P=PRD(U,V-1); Q=PRD(U,V); CALL ENTER(P,Q,'=');
                    IF P<=MM THEN DO;
                        DO I=1 TO M;
                            IF R(P,I) THEN CALL ENTER(I,Q,'>'); END;
                    IF Q<=MM THEN DO J=1 TO M;
```

```
                    IF L(Q,J) THEN DO;
                        CALL ENTER(P,J,'<');
                        DO I=1 TO M;
                            IF R(P,I) THEN CALL ENTER(I,J,'>');
                            END;
                        END;
                    END;
                END;
                ELSE IF Q<=MM THEN DO J=1 TO M;
                    IF L(Q,J) THEN CALL ENTER(P,J,'<'); END;
        END UV;
      PUT FILE(OUT) EDIT('PRECEDENCE MATRIX',' ')(PAGE,A,SKIP,A);
      PUT FILE(OUT) EDIT((J/10  DO J=10 TO M BY 10))
          (SKIP,X(6),9(X(10),F(1)));
      DO I=1 TO M;
          PUT FILE(OUT) EDIT(I,' ')(SKIP,F(4),X(1),A);
          DO J=0 TO M BY 10;
             IF M>J+9 THEN U=J+9; ELSE U=M;
             PUT FILE(OUT) EDIT((H(I,K) DO K=J TO U),'.')
                    (11 A);
             END;
          END;
    IF ERRORFLAG THEN PUT FILE(OUT) EDIT
       ('PRECEDENCE VIOLATIONS OCCURRED')(SKIP(2),A);
    ELSE PUT FILE(OUT) EDIT('NO PRECEDENCE VIOLATIONS OCCURRED')
         (SKIP(2),A);
    END PRECEDENCE;
OUTPUT_DCL:  PROC;
    /*OUTPUT DECLARATIONS TO PARSER FILE*/
    DCL (I,J,K) FIXED BIN;
       PUT FILE(PARSER) EDIT('DCL /*DECLARATIONS FROM SYNTAX*/')
           (COLUMN(6),A);
       IF QUOTES='''' THEN QUOTES=QUOTES||QUOTES;
      PUT FILE(PARSER) EDIT ('QUOTES EXT CHAR(20) VAR INITIAL( ''',
          QUOTES,''')')(COLUMN(10), 3 A);
       PUT FILE(PARSER) EDIT ('ERRORSCAN CHAR(20) VAR INITIAL( ''',
            ERRORSCAN,''')')(COLUMN(10),3 A);
       PUT FILE(PARSER) EDIT ('SCAN_START CHAR(20) VAR INITIAL( '''
           ,SCAN_START,''')')(COLUMN(10),3 A);
       /* MAKE UP BASSYM AND BASVAL  */
       J=0;  XWORD=0;  XINTEGER=0;    XSTRING=0;
       DO I=MM+1 TO M;
           IF SYT(I) = WORD THEN XWORD=I;
           ELSE IF SYT(I)=INTEGER THEN XINTEGER=I;
           ELSE IF SYT(I)=STRING THEN XSTRING=I;
           ELSE DO;
                J=J+1;
                BASSYM(J)=SYT(I);
                BASVAL(J)=I;
                END;
           END;
       J=J+1;
       BASSYM(J)=TERMINAL;
       BASVAL(J)=M+1;
       XTERM=M+1;
           PUT FILE(PARSER) EDIT('BASSYM(',J,') CHAR(20) VAR ',
                 'INITIAL(''',BASSYM(1),''''')(COLUMN(10),  A,
                   F(4), 4 A);
           PUT FILE(PARSER) EDIT ((',''',BASSYM(I),''''  DO I=2 TO
                   J))(COLUMN(20),6 A);
```

```
      PUT FILE(PARSER) EDIT (')',')')(A);
      PUT FILE(PARSER) EDIT ('BASVAL(',J,
        ') FIXED BIN INITIAL(',BASVAL(1))(COLUMN(10),A,F(4),
            A,F(4));
      PUT FILE(PARSER) EDIT ((',',BASVAL(I)  DO I=2 TO J))
           (COLUMN(20), 10(A,F(4)));
      PUT FILE(PARSER) EDIT(')',')')(A);
      PUT FILE(PARSER) EDIT('KEY(0:',M+1,')FIXED BIN INITIAL('
           ,KEY(0))(COLUMN(10),A,F(4),A,F(4));
      PUT FILE(PARSER) EDIT ((',',KEY(I)  DO I=1 TO M+1))
           (COLUMN(20),6(A,F(4)));
      PUT FILE(PARSER) EDIT(')',')')(A);
      PUT FILE(PARSER) EDIT('PRTB(0:',KEY(M+1),
         ') FIXED BIN INITIAL(',PRTB(0))(COLUMN(10),A,F(4),
              A,F(4));
      PUT FILE(PARSER) EDIT ((',',PRTB(I)  DO I=1 TO KEY(M+1)
        ))(COLUMN(20),6(A,F(4)));
      PUT FILE(PARSER) EDIT (')',')')(A);
      PUT FILE(PARSER) EDIT('HLIM FIXED BIN INITIAL (',M+1,
          '),')(COLUMN(10),A,F(4),A);
      PUT FILE(PARSER) EDIT('XTERM FIXED BIN INITIAL(',XTERM,
          '),')(COLUMN(10),A,F(4),A);
      XSEQ,XSCAN_STOP=0; DO K=1 TO 10; XSYM(K)=0; END;
      DO I=1 TO M;
         DO K=1 TO 10;
            IF SYT(I)=SYM(K) THEN XSYM(K)=I;
            END;
         IF SYT(I)=SEQUENCE THEN XSEQ=I;
         ELSE IF SYT(I)=SCAN_STOP THEN XSCAN_STOP=I;
         END;
      PUT FILE(PARSER) EDIT ('XSYM(10) FIXED BIN INITIAL(',
          XSYM(1),(',',XSYM(K) DO K=2 TO 10),')')
          (COLUMN(10),A,F(4),COLUMN(20),9(A,F(4)),A);
      PUT FILE(PARSER) EDIT('XWORD FIXED BIN INITIAL(',XWORD,
            '),')(COLUMN(10),A,F(4),A);
      PUT FILE(PARSER) EDIT('XINTEGER FIXED BIN INITIAL(',
            XINTEGER,')',')(COLUMN(10),A,F(4),A);
      PUT FILE(PARSER) EDIT('XSTRING FIXED BIN INITIAL(',
           XSTRING,')',')(COL(10),A,F(4),A);
      PUT FILE(PARSER) EDIT('XSCAN_STOP FIXED BIN INITIAL(',
            XSCAN_STOP,')',')(COLUMN(10),A,F(4),A);
      PUT FILE(PARSER) EDIT ('XSEQ FIXED BIN INITIAL(',XSEQ,
            '),')(COLUMN(10),A,F(4),A);
      PUT FILE (PARSER) EDIT('M FIXED BIN INITIAL(',J,')',')
           (COLUMN(10),A,F(4),A);
      PUT FILE(PARSER) EDIT('N FIXED BIN INITIAL(',N,');')
           (COLUMN(10),A,F(4),A);
/*     SET UP TO OUTPUT PRECEDENCE INITIALIZING PROCEDURE
       AND PRECEDENCE MATRIX H      */
      PUT FILE(PARSER) EDIT('DCL HINITIAL ENTRY(FIXED BIN);',
          'HINITIAL:   PROC(JLIM);',
          'DCL (I,K,JLIM) FIXED BIN,',
          'J(0:JLIM) FIXED BIN INITIAL(0')
          (COL(10),A,COL(2),A,2(COL(10),A));
/* 0 MEANS =  1 MEANS < AND 2 MEANS >   */
      J=0;
      DO I=1 TO M;
         DO K=1 TO M;
            IF H(I,K)¬=' ' THEN DO;
                PUT FILE(PARSER) EDIT(',',I,',',K,',')
```

```
                        (COL(20),2(A,F(4)),A);
                    IF H(I,K)='=' THEN
                        PUT FILE(PARSER) EDIT('0')(A);
                    ELSE IF H(I,K)='<' THEN
                        PUT FILE(PARSER) EDIT('1')(A);
                    ELSE PUT FILE(PARSER) EDIT('2')(A);
                    J=J+3;
                    END;
                END;
            END;
        PUT FILE(PARSER) EDIT(');')(A);
        PUT FILE(PARSER) EDIT('DO I=0 TO HLIM;',
            'DO K=0 TO HLIM;','H(I,K)='' '';',
            'END;','END;','DO I=1 TO JLIM-1 BY 3;',
            'IF   J(I+2) =0 THEN H(J(I),J(I+1))=''='';',
            'ELSE IF   J(I+2) =1 THEN H(J(I),J(I+1))=''<'';',
            'ELSE H(J(I),J(I+1))=''>'';',
            'END;',
            'DO I=0 TO HLIM;','H(HLIM,I)=''<'';',
            'H(I,HLIM)=''>'';', 'END;',' END HINITIAL;')
            (COL(14),A,COL(18),A,COL(22),A,COL(22),A,
                COL(18),A,COL(14),A,4(COL(18),A),COL(14),A,
              3(COL(18),A),COL(14),A);
        PUT FILE(PARSER) EDIT('DCL H(0:',M+1,',0:',M+1,
            ') CHAR(1) INITIAL CALL HINITIAL(',J,');')
            (COL(10),3(A,F(4)),A);
    END OUTPUT_DCL;
OUTPUT_PARSER:   PROC;
    /*  MERGES INPUT FILE2 WITH DECLARATIONS FROM OUTPUT_DCL INTO
        FILE3 */
    DCL A CHAR(80) VAR;
    DCL I FIXED BIN, (B,C) BIT(1);
        ON ENDFILE(IN) BEGIN;
           PUT FILE(OUT) EDIT('*****ENDFILE PARSER INPUT-*END* ',
                'ABSENT OR WRONG')(SKIP,2 A);
           GO TO EXIT;
           END;
        OPEN FILE(IN) TITLE(FILE2) INPUT STREAM;
        OPEN FILE(PARSER) TITLE(FILE3) OUTPUT STREAM;
        IF PARSER_NAME='' THEN B='0'B; ELSE B='1'B;
        IF SEMANT_NAME='' THEN C='0'B; ELSE C='1'B;
LOOP:   GET FILE(IN) EDIT(A)(A(80));
        IF SUBSTR(A,1,5)='*END*' THEN GO TO EXIT;
        IF SUBSTR(A,1,8)='*INSERT*' THEN CALL OUTPUT_DCL;
            ELSE IF B & INDEX(A,'*PARSER*')¬=0 THEN DO;
               I=INDEX(A,'*PARSER*');
               A=SUBSTR(A,1,I-1)||PARSER_NAME||SUBSTR(A,I+8);
               PUT FILE(PARSER) EDIT(A)(SKIP,A);
               END;
            ELSE IF C & INDEX(A,'*SEMANT*')¬=0  THEN DO;
               I=INDEX(A,'*SEMANT*');
               A=SUBSTR(A,1,I-1)||SEMANT_NAME||SUBSTR(A,I+8);
               PUT FILE(PARSER) EDIT(A)(SKIP,A);
               END;
            ELSE PUT FILE(PARSER) EDIT(A)(SKIP,A);
        GO TO LOOP;
EXIT:   CLOSE FILE(IN);
        CLOSE FILE(PARSER);
        END OUTPUT_PARSER;
         /*----------CALLING SEQUENCE ----------*/
           OPEN FILE(OUT) TITLE(FILE4) PRINT STREAM;
           CALL READ_SYNTAX_INPUT;
           CALL BASIC;
           CALL COMP_KEY_PRTB;
           CALL SYNTAX_OUTPUT;
           CALL PRECEDENCE;
           CALL OUTPUT_PARSER;
           CLOSE FILE(OUT);
         END SYNTAX;
```

APPENDIX C

SKELETON PARSER

```
*PARSER*:  PROC OPTIONS(MAIN);
        /*PARSER USING THE TABLES INSERTED BY THE SYNTAX PROGRAM      */
       DCL  INPUTT CHAR(7) VAR, /*INPUT FILE */
            POUT CHAR(7) VAR,  /*DIAGONOSTIC OUTPUT FILE*/
             OUTPUT CHAR(7) VAR; /*OUTPUT FILE*/
          DCL (I,J,K,L,KK,I1,I2,I3) FIXED BIN,
            S(0:50) FIXED BINARY, /*PARSING STACK*/
            V(0:50) CHAR(400) VAR, /* VALUE STACK */
            QUOTE BIT(1),  /*BOOLEAN FOR QUOTING BASIC SYMBOLS */
            SYM FIXED BIN,  /* NUMERICAL FORM OF ASSIGNED SYMBOL */
            SYMS CHAR(400) VAR, /*STRING FORM OF ASSIGNED SYMBOL */
            ERROR BIT(1) INITIAL('0'B), /*PASSED TO SEMANT*/
            ANS FIXED BIN INITIAL(0), /*PASSED TO SEMANT*/
            INPUT CHAR(100) VAR; /*INPUT BUFFER*/
*INSERT*
        DCL LCOK INTERNAL ENTRY(CHAR(400) VAR,FIXED BIN,BIT(1),BIT(1));
LOOK:   PROC(S,I,T,X);
        /*FREE FIELD READ PROCEDURE  T IS FALSE IF INTEGER ELSE TRUE*/
        /*SEPARATOR IS ALWAYS BLANK IF NOT QUOTED STRING THEN A
          SEPARATOR IS ANY SINGLE CHARACTER IN THE SYNTAX
          IF X TRUE THEN BLANKS REMOVED ELSE BLANKS LEFT */
NEXT:   PROC  RETURNS(CHAR(1));
        /* GETS THE NEXT CHARACTER FROM INPUT*/
           ON ENDFILE (IN) BEGIN;
              PUT FILE(DIAG) LIST('*****ENDFILE MAIN SCANNER') SKIP;
              IF QUOTE THEN PUT FILE(DIAG) LIST
                   ('*****MISMATCHING QUOTES') SKIP;
              GO TO FINIS;
              END;
           IF I>LENGTH(INPUT) THEN DO;
              GET FILE(IN) EDIT(INPUT)(A(80));
              PUT FILE(DIAG) EDIT('NEW INPUT STRING***'  ,INPUT)
                    (SKIP,2 A);
              INPUT = INPUT || ' ';
              I=1;
              END;
           RETURN(SUBSTR(INPUT,I,1));
           END NEXT;
CON:    PROC;
        /*CONCATENATES SYM TO S AND INCREASED I */
           S=S ||SYM; I=I+1;.
           END CON;
SPEC:   PROC(A,B)  RETURNS(BIT(1));
        /* TRUE IF A IS NOT A SEPARATING CHARACTER*/
        DCL A CHAR(1), B BIT(1), J FIXED BIN;
           IF A=' ' | A=QUOTES THEN RETURN('0'B);
           IF B THEN RETURN('1'B);
           DO J=1 TO M;  IF A=BASSYM(J) THEN RETURN('0'B); END;
           RETURN('1'B);
           END SPEC;
        DCL SPEC INTERNAL ENTRY (CHAR(1),BIT(1)) RETURNS(BIT(1)),
            NEXT INTERNAL ENTRY RETURNS (CHAR(1)),
            CON INTERNAL ENTRY,
            SYM CHAR(1),
            (T,X) BIT(1),
```

```
          I FIXED BIN, /*INPUT BUFFER POINTER*/
          S CHAR(400) VAR; /*OUTPUT STRING*/
          SYM=NEXT;   S='';
          IF X THEN DO WHILE (SYM=' ');
             I=I+1;  SYM=NEXT;  END;
          IF ¬SPEC(SYM,QUOTE) THEN DO;
             CALL CON;  T='1'B; RETURN; END;
          IF SYM>'Z' THEN DO;
             DO WHILE (NEXT>'Z');
                CALL CON;   SYM=NEXT;
                END;
             T='0'B;  RETURN;
             END;
          DO WHILE (SPEC(SYM,QUOTE));
             CALL CON;   SYM=NEXT;
             END;
          T='1'B;   RETURN;
          END LOOK;
ASSIGN: PROC (QUOTE,OS,V)  RECURSIVE;
        /*ASSIGNS A NUMERICAL VALUE TO CURRENT INPUT SYMBOL */
        DCL QUOTE BIT(1),
            OS CHAR(400) VAR, /*STRING RETURNED HERE */
            V FIXED BIN,  /* NUMERICAL FORM OF STRING */
            J FIXED BIN,
            T BIT(1),OX CHAR(400) VAR;
            IF QUOTE THEN DO;
               CALL LOOK(OS,I,T,'0'B);
               IF OS=QUOTES THEN DO;
                  QUOTE='0'B;   OS='';  V=XSTRING;        RETURN;
                  END;
               CALL LOOK(OX,I,T,'0'B);
               DO WHILE(OX¬=QUOTES);
                  OS=OS||OX;
                  CALL LOOK(OX,I,T,'0'B);
                  END;
               QUOTE='0'B;  V=XSTRING;
               RETURN;
               END;
            CALL LOOK(OS,I,T,'1'B);
            IF T THEN DO;
               IF OS=QUOTES THEN DO;
                  QUOTE='1'B; CALL ASSIGN(QUOTE,OS,V);  RETURN;
                  END;
               DO J=1 TO M;
                  IF OS=BASSYM(J) THEN DO;
                     V=BASVAL(J); RETURN;
                     END;
                  END;
               V=XWORD;  RETURN;
               END;
             V=XINTEGER;  RETURN;
             END ASSIGN;
 SCAN2:  PROC;
         /* DRAINS INPUT BUFFER AND SCANS INPUT FILE UNTIL SCAN_START
            OCCURS RESET I AND INPUT BUFFER */
         DCL K FIXED BIN;
            ON ENDFILE(IN) BEGIN;
               PUT FILE(DIAG) EDIT('*****ENDFILE ALTERNATE SCANNER',
               '*****CHECK FOR MATCHING SCAN_STOP & SCAN_START')
               (2(SKIP,A));
```

```
            GO TO FINIS;
            END;
         IF I<LENGTH(INPUT) THEN INPUT=' '||SUBSTR(INPUT,I);
            ELSE DO;
               GET FILE(IN) EDIT(INPUT)(A(80));
               PUT FILE(DIAG) EDIT('CODE INPUT STRING**',INPUT)
                   (SKIP,2 A);
               END;
LOOP:    K=INDEX(INPUT,SCAN_START);
         IF K=0 THEN DO;
            PUT FILE(OUT) EDIT(INPUT)(SKIP,A);
            GET FILE(IN) EDIT(INPUT)(A(80));
            PUT FILE(DIAG) EDIT('CODE INPUT STRING**',INPUT)
                (SKIP,2 A);
            GO TO LOOP;
            END;
         IF K+LENGTH(SCAN_START)>=LENGTH(INPUT) THEN DO;
            I=1;  INPUT='';
            END;
         ELSE DO;
            PUT FILE(OUT) EDIT(SUBSTR(INPUT,1,K-1))(COLUMN(2),A);
            I=1;   INPUT=SUBSTR(INPUT,K+LENGTH(SCAN_START));
            END;
         END SCAN2;
STACKOK: PROC  RETURNS(BIT(1));
         /* TRUE IF H(S(J-1),S(J))='<'  */
         DCL I FIXED BIN;
            IF H(S(J-1),S(J))='<' THEN RETURN('1'B);
            PUT FILE(DIAG) LIST('*****ERROR IN PARSING STACK ') SKIP;
            RETURN('0'B);
            END STACKOK;
ERROR_RECOVERY:  PROC;
        /*RESETS STACK, SCANS INPUT UNTIL ERROR_SCAN */
        DCL (M,ER) BIT(1),(R,L,KK) FIXED BIN,(TR,TL,XR,XL) CHAR(400)
           VAR;
        DCL TYPE INTERNAL ENTRY(FIXED BIN) RETURNS (CHAR(400) VAR);
TYPE:   PROC(R)  RETURNS(CHAR(400) VAR);
        /*RETURNS TYPE OF R INTEGER,WORD,STRING OR RESERVED */
        DCL R FIXED BIN;
           IF R=XWORD THEN RETURN('WORD');
           ELSE IF R=XINTEGER THEN RETURN('INTEGER');
           ELSE IF R=XSTRING THEN RETURN('STRING');
           ELSE RETURN('RESERVED WORD');
           END TYPE;
        /*RESET STACK  ---------------- */
           PUT FILE(DIAG) EDIT('*****SYNTAX ANALYSIS I=',I)
               (SKIP,A,F(4));
           PUT FILE(DIAG) EDIT('STACK WAS ',(S(L), V(L) DO L=0 TO K))
               (COLUMN(20),A,5(COLUMN(20),10(F(4),X(1),A)));
           L=1;
           DO WHILE (XSYM(L)¬=0 & L<10);
              L=L+1;
              END;
           M='0'B;   J=L;
           DO KK=1 TO L;
              IF S(KK)¬=XSYM(KK) THEN DO;
                J=KK-1; GO TO EXIT;
                END;
              END;
   EXIT:   IF J=L & ERRORSCAN¬=SCAN_START THEN M='1'B;
```

```
        /* SCAN INPUT UNTIL ERRORSCAN  FOR ERRORS */
           I=1;  QUOTE='0'B;  ER='1'B;
           CALL ASSIGN(QUOTE,XR,R);
           TR=TYPE(R);
LOOP:      IF XR=ERRORSCAN  THEN GO TO XEXIT;
           TL=TR;  XL=XR;   L=R;
           IF L=XSCAN_STOP THEN DO;
              CALL SCAN2;
              IF ERRORSCAN=SCAN_START THEN GO TO XEXIT;
              END;
           CALL ASSIGN(QUOTE,XR,R);
           TR=TYPE(R);
           IF H(L,R)=' ' THEN DO;
              ER='0'B;
              PUT FILE(DIAG) EDIT (XL,'(TYPE-',TL,') MAY NOT BE ',
                 'FOLLOWED BY ',XR,'(TYPE-',TR,')')(COLUMN(20),9 A);
              END;
           GO TO LOOP;
XEXIT:     IF ER THEN PUT FILE(DIAG) EDIT('ERROR NOT IN CURRENT INPUT')
                (COLUMN(20),A);
           PUT FILE(DIAG) EDIT('STACK RESET TO ',(S(L), V(L) DO L=0
              TO J))(COLUMN(10),A,5(COLUMN(20),10(F(4),X(1),A)));
           PUT FILE(DIAG) LIST('*****END OF ANALYSIS') SKIP;
           QUOTE='0'B;
           INPUT=SUBSTR(INPUT,I);     I=1;
           IF M THEN DO;  SYMS=XR;  SYM=R;      END;
             ELSE CALL ASSIGN(QUOTE,SYMS,SYM);
           END ERROR_RECOVERY;
        /* ---------     PARSING SECTION     ------------  */
        DCL STACKOK INTERNAL ENTRY RETURNS(BIT(1));
         DO J=0 TO 50;  S(J)=0;  V(J)= '';   END;
         S(0)=XTERM;
         INPUTT='SOURCE'; POUT='DIAG'; OUTPUT='OUTPUT';
         OPEN FILE(OUT) TITLE(OUTPUT) OUTPUT STREAM;
         OPEN FILE(DIAG) TITLE(POUT) PRINT STREAM;
         OPEN FILE(IN) TITLE (INPUTT) INPUT STREAM;
         I=1;    INPUT='';      J=0;    QUOTE='0'B;
         CALL ASSIGN(QUOTE,SYMS,SYM);
         DO WHILE (SYM>0);
           J=J+1;  K=J;  S(J)=SYM;  V(J)=SYMS;
           IF S(J)=XSCAN_STOP THEN CALL SCAN2;
           CALL ASSIGN(QUOTE,SYMS,SYM);
           DO WHILE (H(S(J),SYM)='>');
              IF S(J)=XSEQ THEN GO TO FINIS;
              DO WHILE ((H(S(J-1),S(J))='=') &(J>1));
                 J=J-1;
                 END;
              L=KEY(S(J));
              IF STACKOK THEN DO WHILE (PRTB(L)¬=0);
                 KK=J+1;
                 DO WHILE ((KK<=K) & (S(KK)=PRTB(L)));
                    KK=KK+1;  L=L+1;
                    END;
                 IF ((KK>K) & (PRTB(L)<0)) THEN DO;
                    I1=J;  I2=K;  I3=-PRTB(L);
                    IF I3<=N THEN CALL *SEMANT*(I3,V,I1,I2,ANS,ERROR);
                    S(J)=PRTB(L+1);    L=0;
                    END;
                 ELSE DO;
                    DO WHILE (PRTB(L)>0);
```

```
                    L=L+1;
                    END;
                 L=L+2;
                 END;
              END;
           ELSE DO;                        /*ELSE TO IF--DO(PRTB--) */
                CALL ERROR_RECOVERY;  L=0;
                END;
           IF L¬=0 THEN DO;          /*PUT ERROR RECOVERY HERE */
               L=0;  CALL ERROR_RECOVERY;
               END;
           K=J;
           END;
        END;
      IF SYM=0 THEN DO;
         PUT FILE(DIAG) LIST
           ('*****THE SYMBOL ',SYMS,' WAS ASSIGNED TO NULL CLASS ')
           SKIP;
         IF XWORD=0 THEN PUT FILE(DIAG) LIST('WORD CLASS ');
         IF XINTEGER=0 THEN PUT FILE(DIAG) LIST('INTEGER CLASS ');
         IF XSTRING=0 THEN PUT FILE(DIAG) LIST('STRING CLASS');
         END;
FINIS:   END *PARSER*;
*END*
```

APPENDIX D -- SEMANTIC CONSTRUCTOR

SYNTAX

```
*SYNTAX*
SEMANTICS *::=* SEMANT  CODA  PRODUCTIONS *;*
PRODUCTIONS *::=* INTERPRETATIONS *NO-SEMANT* *;*
SEMANT *::=* *SEMANTICS* WORD *;*
INTERPRETATIONS *::=* INTERPRETATION *NO-SEMANT* *;*
*::=* INTERPRETATIONS INTERPRETATION *NO-SEMANT* *;*
INTERPRETATION *::=* INTERP *CODE* *;*
INTERP *::=* *PRODUCTION* INTEGER     *;*
CODA *::=* *CODE*
*END-SYNTAX*
```

APPENDIX D -- SEMANTIC CONSTRUCTOR

PARSER WITH SEMANTICS

```
*PARSER*:  PROC (INPUTT,OUTPUT,POUT);
         /*PARSER USING THE TABLES INSERTED BY THE SYNTAX PROGRAM      */
       DCL  INPUTT CHAR(7) VAR, /*INPUT FILE */
            POUT CHAR(7) VAR,  /*DIAGONOSTIC OUTPUT FILE*/
             OUTPUT CHAR(7) VAR, /*OUTPUT FILE*/
             LCOK INTERNAL ENTRY(CHAR(400) VAR,FIXED BIN,BIT(1),BIT(1));
CODE_OUT:  PROC(N,VS,J,K,ANS,ERROR);
          DCL (N,J,K,ANS) FIXED BIN, I FIXED BIN,
               VS(0:50) CHAR(400) VAR, ERROR BIT(1);
             IF N=1 THEN DO;
                PUT FILE(OUT) EDIT ('END '||VS(J)||';')(COL(10),A);
                CLOSE FILE(OUT);
                END;
             ELSE IF N=3 THEN DO;
                PUT FILE(OUT) EDIT
                 (VS(J+1)||': PROC(N,VS,J,K,ANS,ERROR);')(COLUMN(2),A);
                PUT FILE(OUT) EDIT(
                   'DCL N FIXED BIN, /*PRODUCTION NUMBER*/',
                   'VS(0:50) CHAR(400) VAR, /*VALUE STACK */',
                   'J FIXED BIN, /*LEFT STACK POINTER*/',
                   'K FIXED BIN, /*RIGHT STACK POINTER */',
                   'ANS FIXED BIN, /*NOT USED BY PARSER INIT TO 0*/',
                   'ERROR BIT(1); /*NOT USED BY PARSER INIT TO FALSE*/')
                   (COL(10),A,5(COL(14),A));
                VS(J)=VS(J+1);
                END;
             ELSE IF N=6 THEN PUT FILE(OUT) EDIT('RETURN;','END '
                ||'L'||VS(J)||';')(2(COLUMN(10),A));
             ELSE IF N=7 THEN DO;
                PUT FILE(OUT) EDIT('IF N=',VS(K),' THEN','L'||VS(K)||':'
                  ,'DO;                  /*PRODUCTION NUMBER ',VS(K),'*/')
                  (COLUMN(10),3 A,COLUMN(2),A,COLUMN(20),3 A);
                VS(J)=VS(K);
                END;
           END CODE_OUT;
          DCL (I,J,K,L,KK,I1,I2,I3) FIXED BIN,
              S(0:50) FIXED BINARY, /*PARSING STACK*/
              V(0:50) CHAR(400) VAR, /* VALUE STACK */
              QUOTE BIT(1),  /*BOOLEAN FOR QUOTING BASIC SYMBOLS */
              SYM FIXED BIN,  /* NUMERICAL FORM OF ASSIGNED SYMBOL */
              SYMS CHAR(400) VAR, /*STRING FORM OF ASSIGNED SYMBOL */
              ERROR BIT(1) INITIAL('0'B), /*PASSED TO SEMANT*/
              ANS FIXED BIN INITIAL(0), /*PASSED TO SEMANT*/
              INPUT CHAR(100) VAR; /*INPUT BUFFER*/
*INSERT*
 LOOK:    PROC(S,I,T,X);
          /*FREE FIELD READ PROCEDURE T IS FALSE IF INTEGER ELSE TRUE*/
          /*SEPARATOR IS ALWAYS BLANK IF NOT QUOTED STRING THEN A
            SEPARATOR IS ANY SINGLE CHARACTER IN THE SYNTAX
            IF X TRUE THEN BLANKS REMOVED IF FALSE THEN BLANKS LEFT */
 NEXT:    PROC  RETURNS(CHAR(1));
          /* GETS THE NEXT CHARACTER FROM INPUT*/
             ON ENDFILE (IN) BEGIN;
                PUT FILE(DIAG) LIST('*****ENDFILE MAIN SCANNER') SKIP;
                GO TO FINIS;
```

```
            END;
         IF I>LENGTH(INPUT) THEN DO;
            GET FILE(IN) EDIT(INPUT)(A(80));
            PUT FILE(DIAG) EDIT('NEW INPUT STRING***'  ,INPUT)
                  (SKIP,2 A);
            INPUT = INPUT || ' ';
            I=1;
            END;
         RETURN(SUBSTR(INPUT,I,1));
         END NEXT;
CON:     PROC;
         /*CONCATENATES SYM TO S AND INCREASED I */
            S=S ||SYM; I=I+1;
            END CON;
SPEC:    PROC(A,B)  RETURNS(BIT(1));
         /* TRUE IF A IS NOT A SEPARATING CHARACTER*/
         DCL A CHAR(1), B BIT(1), J FIXED BIN;
            IF A=' ' | A=QUOTES THEN RETURN('0'B);
            IF B THEN RETURN('1'B);
            DO J=1 TO M;  IF A=BASSYM(J) THEN RETURN('0'B); END;
            RETURN('1'B);
            END SPEC;
         DCL SPEC INTERNAL ENTRY (CHAR(1),BIT(1)) RETURNS(BIT(1)),
             NEXT INTERNAL ENTRY RETURNS (CHAR(1)),
             CON INTERNAL ENTRY,
             SYM CHAR(1),
             (T,X) BIT(1),
             I FIXED BIN, /*INPUT BUFFER POINTER*/
             S CHAR(400) VAR; /*OUTPUT STRING*/
             SYM=NEXT;   S='';
             IF X THEN DO WHILE (SYM=' ');
                I=I+1;  SYM=NEXT;  END;
             IF ¬SPEC(SYM,QUOTE) THEN DO;
                CALL CON;  T='1'B; RETURN; END;
             IF SYM>'Z' THEN DO;
                DO WHILE (NEXT>'Z');
                   CALL CON;   SYM=NEXT;
                   END;
                T='0'B;  RETURN;
                END;
             DO WHILE (SPEC(SYM,QUOTE));
                CALL CON;   SYM=NEXT;
                END;
             T='1'B;   RETURN;
             END LOOK;
ASSIGN:  PROC (QUOTE,OS,V)  RECURSIVE;
         /*ASSIGNS A NUMERICAL VALUE TO CURRENT INPUT SYMBOL */
         DCL QUOTE BIT(1),
             OS CHAR(400) VAR, /*STRING RETURNED HERE */
             V FIXED BIN,  /* NUMERICAL FORM OF STRING */
             J FIXED BIN,
             T BIT(1),OX CHAR(400) VAR;
             IF QUOTE THEN DO;
                CALL LOOK(OS,I,T,'0'B);
                IF OS=QUOTES THEN DO;
                   QUOTE='0'B;   OS='';  V=XSTRING;       RETURN;
                   END;
                CALL LOOK(OX,I,T,'0'B);
                DO WHILE (OX¬=QUOTES);
                   OS=OS||OX;
```

```
                  CALL LOOK(OX,I,T,'0'B);
                  END;
               QUOTE='0'B;  V=XSTRING;
               RETURN;
               END;
            CALL LOOK(OS,I,T,'1'B);
            IF T THEN DO;
               IF OS=QUOTES THEN DO;
                  QUOTE='1'B; CALL ASSIGN(QUOTE,OS,V);  RETURN;
                  END;
               DO J=1 TO M;
                  IF OS=BASSYM(J) THEN DO;
                     V=BASVAL(J); RETURN;
                     END;
                  END;
               V=XWORD;  RETURN;
               END;
            V=XINTEGER;  RETURN;
            END ASSIGN;
SCAN2:   PROC;
         /* DRAINS INPUT BUFFER AND SCANS INPUT FILE UNTIL SCAN_START
            OCCURS RESET I AND INPUT BUFFER */
         DCL K FIXED BIN;
            ON ENDFILE(IN) BEGIN;
               PUT FILE(DIAG) EDIT('*****ENDFILE ALTERNATE SCANNER',
               '*****CHECK FOR MATCHING SCAN_STOP & SCAN_START')
               (2(SKIP,A));
               GO TO FINIS;
               END;
            IF I<LENGTH(INPUT) THEN INPUT=' '||SUBSTR(INPUT,I);
               ELSE DO;
                  GET FILE(IN) EDIT(INPUT)(A(80));
                  PUT FILE(DIAG) EDIT('CODE INPUT STRING**',INPUT)
                       (SKIP,2 A);
                  END;
LOOP:       K=INDEX(INPUT,SCAN_START);
            IF K=0 THEN DO;
               PUT FILE(OUT) EDIT(INPUT)(SKIP,A);
               GET FILE(IN) EDIT(INPUT)(A(80));
               PUT FILE(DIAG) EDIT('CODE INPUT STRING**',INPUT)
                  (SKIP,2 A);
               GO TO LOOP;
               END;
            IF K+LENGTH(SCAN_START)>=LENGTH(INPUT) THEN DO;
               I=1;  INPUT='';
               END;
            ELSE DO;
               PUT FILE(OUT) EDIT(SUBSTR(INPUT,1,K-1))(COLUMN(2),A);
               I=1;   INPUT=SUBSTR(INPUT,K+LENGTH(SCAN_START));
               END;
            END SCAN2;
STACKOK: PROC  RETURNS(BIT(1));
         /* TRUE IF H(S(J-1),S(J))='<'  */
         DCL I FIXED BIN;
            IF H(S(J-1),S(J))='<' THEN RETURN('1'B);
            PUT FILE(DIAG) LIST('*****ERROR IN PARSING STACK ') SKIP;
            RETURN('0'B);
            END STACKOK;
ERROR_RECOVERY:  PROC;
        /*RESETS STACK, SCANS INPUT UNTIL ERROR_SCAN */
```

```
        DCL (M,ER) BIT(1),(R,L,KK) FIXED BIN,(TR,TL,XR,XL) CHAR(400)
           VAR;
        DCL TYPE INTERNAL ENTRY(FIXED BIN) RETURNS (CHAR(400) VAR);
TYPE:   PROC(R)  RETURNS(CHAR(400) VAR);
        /*RETURNS TYPE OF R INTEGER,WORD OR RESERVED */
        DCL R FIXED BIN;
           IF R=XWORD THEN RETURN('WORD');
           ELSE IF R=XINTEGER THEN RETURN('INTEGER');
           ELSE IF R=XSTRING THEN RETURN('STRING');
           ELSE RETURN('RESERVED WORD');
           END TYPE;
        /*RESET STACK  ----------------- */
           PUT FILE(DIAG) EDIT('*****SYNTAX ANALYSIS I=',I)
               (SKIP,A,F(4));
           PUT FILE(DIAG) EDIT('STACK WAS ',(S(L), V(L) DO L=0 TO K))
               (COLUMN(20),A,5(COLUMN(20),10(F(4),X(1),A)));
           L=1;
           DO WHILE (XSYM(L)¬=0 & L<10);
              L=L+1;
              END;
           M='0'B;   J=L;
           DO KK=1 TO L;
              IF S(KK)¬=XSYM(KK) THEN DO;
                J=KK-1; GO TO EXIT;
                END;
              END;
   EXIT:   IF J=L & ERRORSCAN¬=SCAN_START THEN M='1'B;
        /* SCAN INPUT UNTIL ERRORSCAN  FOR ERRORS */
           I=1;  QUOTE='0'B;  ER='1'B;
           CALL ASSIGN(QUOTE,XR,R);
           TR=TYPE(R);
LOOP:      IF XR=ERRORSCAN  THEN GO TO XEXIT;
           TL=TR;  XL=XR;   L=R;
           IF L=XSCAN_STOP THEN DO;
              CALL SCAN2;
              IF ERRORSCAN=SCAN_START THEN GO TO XEXIT;
              END;
           CALL ASSIGN(QUOTE,XR,R);
           TR=TYPE(R);
           IF H(L,R)=' ' THEN DO;
              ER='0'B;
              PUT FILE(DIAG) EDIT (XL,'(TYPE-',TL,') MAY NOT BE ',
                 'FOLLOWED BY ',XR,'(TYPE-',TR,')')(COLUMN(20),9 A);
              END;
           GO TO LOOP;
XEXIT:     IF ER THEN PUT FILE(DIAG) EDIT('ERROR NOT IN CURRENT INPUT')
                (COLUMN(20),A);
           PUT FILE(DIAG) EDIT('STACK RESET TO ',(S(L), V(L) DO L=0
              TO J))(COLUMN(10),A,5(COLUMN(20),10(F(4),X(1),A)));
           PUT FILE(DIAG) LIST('*****END OF ANALYSIS') SKIP;
           QUOTE='0'B;
           INPUT=SUBSTR(INPUT,I);     I=1;
           IF M THEN DO;  SYMS=XR;  SYM=R;      END;
             ELSE CALL ASSIGN(QUOTE,SYMS,SYM);
           END ERROR_RECOVERY;
        /* ---------     PARSING SECTION     ------------  */
        DCL STACKOK INTERNAL ENTRY RETURNS(BIT(1));
         DO J=0 TO 50;  S(J)=0;  V(J)= '';   END;
         S(0)=XTERM;
         OPEN FILE(OUT) TITLE(OUTPUT) OUTPUT STREAM;
```

```
          OPEN FILE(DIAG) TITLE(POUT) PRINT STREAM;
          OPEN FILE(IN) TITLE (INPUTT) INPUT STREAM;
          I=1;    INPUT='';      J=0;    QUOTE='0'B;
          CALL ASSIGN(QUOTE,SYMS,SYM);
          DO WHILE (SYM>0);
            J=J+1;  K=J;  S(J)=SYM;  V(J)=SYMS;
            IF S(J)=XSCAN_STOP THEN CALL SCAN2;
            CALL ASSIGN(QUOTE,SYMS,SYM);
            DO WHILE (H(S(J),SYM)='>');
               IF S(J)=XSEQ THEN GO TO FINIS;
               DO WHILE ((H(S(J-1),S(J))='=') &(J>1));
                  J=J-1;
                  END;
               L=KEY(S(J));
               IF STACKOK THEN DO WHILE (PRTB(L)¬=0);
                  KK=J+1;
                  DO WHILE ((KK<=K) & (S(KK)=PRTB(L)));
                     KK=KK+1;  L=L+1;
                     END;
                  IF ((KK>K) & (PRTB(L)<0)) THEN DO;
                     I1=J;  I2=K;  I3=-PRTB(L);
                     IF I3<=N THEN CALL *SEMANT*(I3,V,I1,I2,ANS,ERROR);
                     S(J)=PRTB(L+1);    L=0;
                     END;
                  ELSE DO;
                     DO WHILE (PRTB(L)>0);
                        L=L+1;
                        END;
                     L=L+2;
                     END;
                  END;
               ELSE DO;                    /*ELSE TO IF--DO(PRTB--) */
                    CALL ERROR_RECOVERY;  L=0;
                    END;
               IF L¬=0 THEN DO;        /*PUT ERROR RECOVERY HERE */
                   L=0;  CALL ERROR_RECOVERY;
                   END;
               K=J;
               END;
            END;
 FINIS:     END *PARSER*;
*END*
```

APPENDIX E -- CONTROL LANGUAGE META SYSTEM

SYNTAX

```
//GO.SYNDATA DD *
 SYM(1)='OPTIONS'       ERRORSCAN='*END*' SEQUENCE='COMMAND-TABLE'
 PARSER_NAME='TABLE'        SEMANT_NAME='SEMANT' QUOTES=''''
 TERMINAL='*END-TABLE*' MLIM=50 NLIM=50 MMLIM=50 SYM(2)='COMMAND-LIST';
/*
//GO.SYNTAX DD *
*SYNTAX*
 COMMAND-TABLE *:::=* OPTIONS COMMAND-LIST* *;*
OPTIONS *:::=* OPTION *NO-SEMANT* *;*
 *:::=* OPTIONS OPTION *NO-SEMANT* *;*
 OPTION *:::=* *QUOTES* *=* WORD *;*
 *:::=* *PERIOD* *=* WORD *;*
 *:::=* *TBL-NAME* *=* STRING *;*
COMMAND-LIST* *:::=* COMMAND-LIST *NO-SEMANT* *;*
COMMAND-LIST *:::=* COMMAND *NO-SEMANT* *;*
 *:::=* COMMAND-LIST COMMAND *NO-SEMANT* *;*
COMMAND *:::=* ID-LIST PARM-LIST* *NO-SEMANT* *;*
ID-LIST *:::=* ID-SPEC *NO-SEMANT* *;*
 *:::=* ID-LIST ID-SPEC *NO-SEMANT* *;*
ID-SPEC *:::=* ID *;*
   *:::=* ID *DL-EX-LIST* STRING *;*
   *:::=* ID *DL-SKIP* STRING *;*
   *:::=* ID *DL-EX-LIST* STRING  *DL-SKIP* STRING *;*
   *:::=* ID *DL-SKIP* STRING *DL-EX-LIST* STRING *;*
ID *:::=* *KEYWORD* WORD *RTN* WORD *;*
 *:::=* *SUB-ENTRY* WORD *;*
PARM-LIST* *:::=* PARM-LIST *NO-SEMANT* *;*
PARM-LIST *:::=* PARM *END* *NO-SEMANT* *;*
 *:::=* PARM-LIST PARM *END* *NO-SEMANT* *;*
PARM *:::=* PARM-ID  *NO-SEMANT* *;*
 *:::=* PARM-ID  KEYS* *NO-SEMANT* *;*
PARM-ID *:::=* *PARM*  TYPE *;*
 *:::=* *PARM*  TYPE *INITIAL* STRING *;*
TYPE *:::=* V-TYPE *;*
     *:::=* V-TYPE P-ACTION *;*
 *:::=* V-TYPE K-REQUIRED *;*
 *:::=* V-TYPE P-ACTION K-REQUIRED *;*
 *:::=* V-TYPE K-REQUIRED P-ACTION *;*
V-TYPE *:::=* *NUM* *NO-SEMANT* *;*
 *:::=* *STRING* *NO-SEMANT* *;*
 *:::=* *NAME* *NO-SEMANT* *;*
 *:::=* STRING   *;*
P-ACTION *:::=* *P* *NO-SEMANT* *;*
K-REQUIRED *:::=* *K* *NO-SEMANT* *;*
KEYS* *:::=* KEYS  *NO-SEMANT* *;*
KEYS *:::=* KEY TYPE-KEY  *;*
 *:::=* KEYS KEY TYPE-KEY  *;*
KEY *:::=* *KEY* WORD *;*
TYPE-KEY *:::=* *VALUE*  *;*
 *:::=* *SELF*  STRING *;*
 *:::=* *VALUE* STRING *;*
 *:::=* *VALUESHORT* STRING   *;*
 *:::=* *CALL* STRING
*END-SYNTAX*
```

APPENDIX E -- CONTROL LANGUAGE META SYSTEM

SEMANTICS

```
//GO.SEMANTIC DD *
*SEMANTICS* SEMANT *CODE*
         DCL I FIXED BIN, TBL(500) EXT CHAR(80) VAR,
             (NAME,NUMBER) INT ENTRY(CHAR(*) VAR) RETURNS(BIT(1)),
             QUOTES EXT CHAR(20) VAR INITIAL(''''),
             PERIOD EXT CHAR(1) INITIAL('.'),
             TBL_NAME EXT CHAR(40) VAR INITIAL('TABLE'),
             TOUT FILE ENVIRONMENT(F(400,80));
 NAME:    PROC(A) RETURNS(BIT(1));
          /*RETURNS TRUE IF A OF TYPE NAME ELSE FALSE */
          DCL A CHAR(*) VAR, J FIXED BIN;
             IF A=' ' | A='' THEN RETURN('0'B);
             IF SUBSTR(A,1,1)<'A' | SUBSTR(A,1,1)>'Z' THEN
                   RETURN('0'B);
             DO J=2 TO LENGTH(A);
                IF SUBSTR(A,J,1)<'A' THEN RETURN('0'B);
                END;
             RETURN('1'B);
             END NAME;
 NUMBER:   PROC(A) RETURNS(BIT(1));
           /*RETURNS TRUE IF A OF TYPE NUMBER ELSE FALSE*/
           DCL A CHAR(*) VAR, X FLOAT BIN;
              ON CONVERSION GO TO FALSE;
              ON OVERFLOW GO TO FALSE;
              ON UNDERFLOW GO TO FALSE;
              X=A;
              RETURN('1'B);
 FALSE:       RETURN('0'B);
              END NUMBER;
           *END*
*PRODUCTION* 1 *CODE*
        /* OUTPUT TABLES */
           OPEN FILE(TOUT) TITLE('TABLES') OUTPUT STREAM;
           TBL(ANS+1)=DATE;
           TBL(ANS+1)=SUBSTR(TBL(ANS+1),3,2)||'/'||
               SUBSTR(TBL(ANS+1),5,2)||'/'||SUBSTR(TBL(ANS+1),1,2);
           TBL(ANS+2)=TIME;
           TBL(ANS+2)=SUBSTR(TBL(ANS+2),1,2)||':'||
                      SUBSTR(TBL(ANS+2),3,2)||':'||
                      SUBSTR(TBL(ANS+2),5,2)||'.'||
                      SUBSTR(TBL(ANS+2),7,3);
           PUT FILE(TOUT) EDIT(TBL_NAME,TBL(ANS+1),TBL(ANS+2),' ')
                     (COL(2),A,X(2),A,X(2),A,SKIP(2),A);
           DO I=1 TO ANS;
              PUT FILE(TOUT) EDIT(TBL(I))(SKIP,A);
              END;
            PUT FILE(TOUT) EDIT('$$$')(SKIP,A);
           *END*
*PRODUCTION* 4 *CODE*
        /* SET QUOTES  */
           QUOTES=VS(K);
           *END*
*PRODUCTION* 5 *CODE*
        /*SET  PERIOD */
           PERIOD=VS(K);
```

```
          *END*
*PRODUCTION* 6 *CODE*
       /* SET TBL_NAME */
          TBL_NAME=VS(K);
          *END*
*PRODUCTION* 13 *CODE*
       /* BUILD ID-SPEC */
          ANS=ANS+1;
          TBL(ANS)=VS(J)||PERIOD||PERIOD||PERIOD;
          *END*
*PRODUCTION* 14 *CODE*
       /* BUILD ID-SPEC WITH EXCL LIST */
          ANS=ANS+1;
          TBL(ANS)=VS(J)||PERIOD||VS(K)||PERIOD||PERIOD;
          *END*
*PRODUCTION* 15 *CODE*
       /* BUILD ID-SPEC WITH SKIP LIST */
          ANS=ANS+1;
          TBL(ANS)=VS(J)||PERIOD||PERIOD||VS(K)||PERIOD;
          *END*
*PRODUCTION* 16 *CODE*
       /* BUILD ID-SPEC WITH EXCL LIST AND SKIP LIST */
          ANS=ANS+1;
          TBL(ANS)=VS(J)||PERIOD||VS(J+2)||PERIOD||VS(K)||PERIOD;
          *END*
*PRODUCTION* 17 *CODE*
       /* BUILD ID-SPEC WITH EXCL LIST AND SKIP LIST */
          ANS=ANS+1;
          TBL(ANS)=VS(J)||PERIOD||VS(K)||PERIOD||VS(J+2)||PERIOD;
          *END*
*PRODUCTION* 18 *CODE*
       /*  SAVE KEYWORD AND RTN */
          VS(J)=VS(J+1)||PERIOD||VS(K);
          *END*
*PRODUCTION* 19 *CODE*
       /*  SAVE ENTRY */
          VS(J)=VS(J+1)||PERIOD;
          *END*
*PRODUCTION* 25 *CODE*
       /*ENTER PARAMETER  AND TYPE*/
          ANS=ANS+1;
          TBL(ANS)=PERIOD||VS(K)||PERIOD||PERIOD;
          VS(J)=VS(K);
          *END*
*PRODUCTION* 26 *CODE*
       /*  ENTER PARAMETER  TYPE, INITIAL VALUE */
       /*  CHECK INITIAL VALUE TYPE  */
          ANS=ANS+1;
          TBL(ANS)=PERIOD||VS(J+1)||PERIOD||VS(K)||PERIOD;
          IF INDEX(VS(J+1),'*NUM*')¬=0 THEN
             IF ¬NUMBER(VS(K)) THEN
                PUT FILE(DIAG) LIST
              ('DIAGNOSTIC MESSAGE*WRONG TYPE INITIAL VALUE') SKIP;
          IF INDEX(VS(J+1),'*NAME*')¬=0 THEN
             IF ¬NAME(VS(K)) THEN
                PUT FILE(DIAG) LIST
              ('DIAGNOSTIC MESSAGE*WRONG TYPE INITIAL VALUE') SKIP;
          *END*
*PRODUCTION* 27 *CODE*
       /* ENTER NULL FOR P K OPTIONS */
```

```
          VS(J)=VS(J)||'**';
          *END*
*PRODUCTION* 28 *CODE*
       /* BUILD TYPE */
          VS(J)=VS(J)||'P**';
          *END*
*PRODUCTION* 29 *CODE*
       /* BUILD TYPE */
          VS(J)=VS(J)||'*K*';
          *END*
*PRODUCTION* 30 *CODE*
       /* BUILD TYPE */
          VS(J)=VS(J)||'P*K*';
          *END*
*PRODUCTION* 31 *CODE*
       /* BUILD TYPE */
          VS(J)=VS(J)||'P*K*';
          *END*
*PRODUCTION* 35 *CODE*
       /* SAVE TYPE WITH * AT END */
          VS(J)=VS(J)||'*';
          *END*
*PRODUCTION* 39 *CODE*
       /*ENTER KEY TYPE-KEY INTO TBL */
          ANS=ANS+1;
          TBL(ANS)=PERIOD||PERIOD||VS(J)||PERIOD||VS(K);
          *END*
*PRODUCTION* 40 *CODE*
       /* ENTER KEY TYPE-KEY INTO TBL */
          ANS=ANS+1;
          TBL(ANS)=PERIOD||PERIOD||VS(J+1)||PERIOD||VS(K);
          *END*
*PRODUCTION* 41 *CODE*
       /* SAVE KEY */
          VS(J)=VS(K);
          *END*
*PRODUCTION* 42 *CODE*
       /*SAVE VALUE */
          VS(J)=VS(J)||PERIOD||PERIOD;
          *END*
*PRODUCTION* 43 *CODE*
       /*SAVE SELF AND STRING */
          VS(J)=VS(J)||PERIOD||VS(K)||PERIOD;
          *END*
*PRODUCTION* 44 *CODE*
       /* SAVE VALUE AND STRING */
          VS(J)=VS(J)||PERIOD||VS(K)||PERIOD;
          *END*
*PRODUCTION* 45 *CODE*
       /* SAVE VALUE AND STRING */
          VS(J)=VS(J)||PERIOD||VS(K)||PERIOD;
          *END*
*PRODUCTION* 46 *CODE*
       /*  SAVE CALL AND STRING */
          VS(J)=VS(J)||PERIOD||VS(K)||PERIOD;
          *END*
*END-SEMANTICS*
```

APPENDIX F -- WYLBUR EXAMPLE

COMMAND DESCRIPTION

```
 *TBL-NAME* *=* 'WYLBUR EXAMPLE---GEORGE'
*QUOTES* *=* @ *PERIOD* *=* :
*SUB-ENTRY* NUMBER
    *PARM* *NUM* *INITIAL* @-1@
        *KEY* FIRST *SELF* @-2@
        *KEY* END *SELF* @-3@
        *KEY* LAST *SELF* @-3@
        *KEY* ALL *SELF* @-4@     *END*
*SUB-ENTRY* NRANGE  *DL-EX-LIST* @,/¬()'"@
    *PARM* @NUMBER @ *K*
        *KEY* , *VALUE* *END*
    *PARM* @NUMBER @ *K* *P*
        *KEY* / *VALUE* *END*
    *PARM* @NRANGE ,@ *K* *P*
        *KEY* , *VALUE* *END*
*SUB-ENTRY* ARANGE *DL-EX-LIST* @'/()"¬,@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ¬ *CALL* @STRINGA ¬@
        *KEY* ' *CALL* @STRINGA '@
        *KEY* " *CALL* @STRINGA "@
        *END*
*SUB-ENTRY* STRINGA *DL-EX-LIST* @'"/()¬,@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ¬ *SELF* @¬@ *END*
    *PARM* *STRING* *K* *P* *INITIAL* @@
        *KEY* ' *CALL* @STRINGB '@
        *KEY* " *CALL* @STRINGB "@
        *END*
*SUB-ENTRY* STRINGB *DL-EX-LIST* @"'/(),@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ' *VALUE* @'@
        *KEY* " *VALUE* @"@  *END*
    *PARM* *NUM* *P* *INITIAL* @-1@ *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* / *VALUE* *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* ( *VALUE* @)@ *END*
*SUB-ENTRY* EQNUM
    *PARM* *NUM*  *INITIAL* @-1@
        *KEY* = *VALUE* *END*
*SUB-ENTRY* STRING *DL-EX-LIST* @'"@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ' *VALUE* @'@
        *KEY* " *VALUE* @"@  *END*
*KEYWORD* LIST *RTN* SUB1 *DL-EX-LIST* @¬'"/(),@
*KEYWORD* L *RTN* SUB1 *DL-EX-LIST* @¬'"/(),@
    *PARM* @ARANGE @ *INITIAL* @@ *END*
    *PARM* @NRANGE ,@ *INITIAL* @@
        *KEY* IN *VALUE*     *END*
*KEYWORD* CHANGE *RTN* SUB2 *DL-EX-LIST* @¬'"/(),@
*KEYWORD* CH *RTN* SUB2 *DL-EX-LIST* @¬'"/(),@
    *PARM* @ARANGE @ *INITIAL* @@ *END*
    *PARM* @STRING @ *K* *P* *INITIAL* @@
        *KEY* TO *VALUE* *END*
    *PARM* @NRANGE ,@ *INITIAL* @@
```

```
        *KEY* IN *VALUE* *END*
*KEYWORD* COPY *RTN* SUB3 *DL-EX-LIST* @,/@
*KEYWORD* CO *RTN* SUB3 *DL-EX-LIST* @,/@
    *PARM* @NRANGE ,@ *INITIAL* @@ *END*
    *PARM* @NUMBER @ *K* *INITIAL* @@
        *KEY* TO *VALUE* *END*
    *PARM* *NUM* *K* *INITIAL* @-1@
        *KEY* BY *VALUE* *END*
*KEYWORD* SET *RTN* SUB4 *DL-EX-LIST* @=@
    *PARM* @EQNUM @ *K* *INITIAL* @@
        *KEY* DELTA *VALUE* *END*
    *PARM* @EQNUM @ *K* *INITIAL* @@
        *KEY* LENGTH *VALUE* *END*
    *PARM* *NUM* *K* *INITIAL* @0@
        *KEY* UPLOW *SELF* @1@
        *KEY* UPPER *SELF* @2@
        *KEY* VERBOSE *SELF* @3@
        *KEY* TERSE *SELF* @4@ *END*
*END-TABLE*
```

APPENDIX F -- WYLBUR EXAMPLE

TABLE

```
 WYLBUR EXAMPLE---GEORGE  07/17/70  14:33:48.260

NUMBER::::
:*NUM***:-1:
::FIRST:*SELF*:-2:
::END:*SELF*:-3:
::LAST:*SELF*:-3:
::ALL:*SELF*:-4:
NRANGE::,/¬()'":
:NUMBER **K*::
::,:*VALUE*::
:NUMBER *P*K*::
::/:*VALUE*::
:NRANGE ,*P*K*::
::,:*VALUE*::
ARANGE::'/()"¬,::
:*STRING**K*::
::¬:*CALL*:STRINGA ¬:
::':*CALL*:STRINGA ':
::":*CALL*:STRINGA ":
STRINGA::'"/()¬,::
:*STRING**K*::
::¬:*SELF*:¬:
:*STRING*P*K*::
::':*CALL*:STRINGB ':
::":*CALL*:STRINGB ":
STRINGB::"'/(),::
:*STRING**K*::
::':*VALUE*:':
::":*VALUE*:":
:*NUM*P**:-1:
:*NUM*P*K*:-1:
::/:*VALUE*::
:*NUM*P*K*:-1:
::(:*VALUE*:):
EQNUM::::
:*NUM***:-1:
::=:*VALUE*::
STRING::'"::
:*STRING**K*::
::':*VALUE*:':
::":*VALUE*:":
LIST:SUB1:¬'"/(),::
L:SUB1:¬'"/(),::
:ARANGE ***::
:NRANGE ,***::
::IN:*VALUE*::
CHANGE:SUB2:¬'"/(),::
CH:SUB2:¬'"/(),::
:ARANGE ***::
:STRING *P*K*::
::TO:*VALUE*::
:NRANGE ,***::
```

```
::IN:*VALUE*::
COPY:SUB3:,/::
CO:SUB3:,/::
:NRANGE ,***::
:NUMBER **K*::
::TO:*VALUE*::
:*NUM**K*:-1:
::BY:*VALUE*::
SET:SUB4:=::
:EQNUM **K*::
::DELTA:*VALUE*::
:EQNUM **K*::
::LENGTH:*VALUE*::
:*NUM**K*:0:
::UPLOW:*SELF*:1:
::UPPER:*SELF*:2:
::VERBOSE:*SELF*:3:
::TERSE:*SELF*:4:
$$$
```

APPENDIX G -- CRBE EXAMPLE

COMMAND DESCRIPTION

```
*TBL-NAME* *=* 'CRBE EXAMPLE---GEORGE'
*QUOTES* *=* @
*PERIOD* *=* :
*SUB-ENTRY* NRANGE *DL-EX-LIST* @'"()¬=@
    *PARM* @BNUM @ *INITIAL* @-1@ *END*
    *PARM* @LNUM @ *P* *INITIAL* @-1@ *END*
    *PARM* @VAL @ *P* *INITIAL* @-1@  *END*
*SUB-ENTRY* VAL *DL-EX-LIST* @'"()=¬@
    *PARM* *NUM* *K* *INITIAL* @-1@
        *KEY* ( *VALUE* @)@ *END*
*SUB-ENTRY* BNUM *DL-EX-LIST* @()'"=¬@
    *PARM* *NUM* *INITIAL* @-1@
        *KEY* FIRST *SELF* @0@ *END*
*SUB-ENTRY* LNUM *DL-EX-LIST* @()¬='"@
    *PARM* *NUM* *INITIAL* @-1@
        *KEY* LAST *SELF* @-2@ *END*
*SUB-ENTRY* ARANGE *DL-EX-LIST* @'"()¬=@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ¬ *CALL* @STRINGA ¬@
        *KEY* ' *CALL* @STRINGA '@
        *KEY* " *CALL* @STRINGA "@
        *END*
*SUB-ENTRY* STRINGA *DL-EX-LIST* @'"()¬=@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ¬ *SELF* @¬@ *END*
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ' *CALL* @STRINGB '@
        *KEY* " *CALL* @STRINGB "@ *END*
*SUB-ENTRY* STRINGB *DL-EX-LIST* @'"()¬=@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ' *VALUE* @'@
        *KEY* " *VALUE* @"@ *END*
    *PARM* *STRING* *K* *P* *INITIAL* @@
        *KEY* COL *CALL* @CB @ *END*
    *PARM* *NUM* *K* *P* *INITIAL* @0@
        *KEY* SEQ *SELF* @0@
        *KEY* NOSEQ *SELF* @1@ *END*
*SUB-ENTRY* COLUMN *DL-EX-LIST* @/'"()=¬@
    *PARM* @CB @ *K* *INITIAL* @@
        *KEY* COL *VALUE* *END*
*SUB-ENTRY* CB *DL-EX-LIST* @'"()=¬@
    *PARM* @CBB @ *K* *INITIAL* @@
        *KEY* = *VALUE* *END*
*SUB-ENTRY* CBB *DL-EX-LIST* @'"(),¬=@
    *PARM* *NUM* *K* *INITIAL* @-1@
        *KEY* ( *VALUESHORT* @),@   *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* , *VALUE* @)@
        *KEY* ) *SELF* @-1@ *END*
*SUB-ENTRY* DSPEC *DL-EX-LIST* @=,().@
    *PARM* @NAM @ *K*
        *KEY* = *VALUE* *END*
    *PARM* *NAME* *K* *P*
        *KEY* ( *VALUE* @)@ *END*
    *PARM* @EQNAM @ *K* *P* *INITIAL* @@
```

```
        *KEY* , *VALUE* *END*
*SUB-ENTRY* EQNAM *DL-EX-LIST* @=()@
    *PARM* @EQNAMB @ *K*
        *KEY* VOL *VALUE*
        *KEY* V *VALUE* *END*
*SUB-ENTRY* EQNAMB *DL-EX-LIST* @=()@
    *PARM* *NAME* *K*
        *KEY* = *VALUE* *END*
*SUB-ENTRY* STRINGC *DL-EX-LIST* @'"()¬@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ¬ *SELF* @¬@     *END*
    *PARM* *STRING* *K* *P* *INITIAL* @@
        *KEY* ' *CALL* @STRINGD '@
        *KEY* " *CALL* @STRINGD "@
        *END*
*SUB-ENTRY* STRINGD *DL-EX-LIST* @'#()¬"@
    *PARM* *STRING* *K* *INITIAL* @@
        *KEY* ' *VALUE* @'@
        *KEY* " *VALUE* @"@  *END*
*SUB-ENTRY* NAM *DL-EX-LIST* @,().@
    *PARM* *NAME*
        *KEY* , *SELF* @ACTIVE@ *END*
    *PARM* @NAMC @ *K*
        *KEY* . *VALUE* *END*
*SUB-ENTRY* NAMC *DL-EX-LIST* @,().@
    *PARM* *NAME* *END*
    *PARM* @NAMC @ *K*
        *KEY* . *VALUE* *END*
*KEYWORD* LIST *RTN* SUB1 *DL-EX-LIST* @()¬'"=@
*KEYWORD* L    *RTN* SUB1 *DL-EX-LIST* @()¬'"=@
    *PARM* @NRANGE @ *INITIAL* @@ *END*
    *PARM* @ARANGE @ *P* *INITIAL* @@ *END*
*KEYWORD* SAVE *RTN* SUB2 *DL-EX-LIST* @.()@
*KEYWORD* S    *RTN* SUB2 *DL-EX-LIST* @.()@
    *PARM* @NAM @  *END*
    *PARM* @CBB (@ *P* *K* *INITIAL* @@
        *KEY* ( *VALUE* *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* KEEP *SELF* @0@
        *KEY* PURGE *SELF* @1@ *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* REPLACE *SELF* @0@
        *KEY* REPL *SELF* @0@ *END*
*KEYWORD* BRING *RTN* SUB3 *DL-EX-LIST* @=()@
*KEYWORD* B    *RTN* SUB3 *DL-EX-LIST* @=()@
    *PARM* @DSPEC @ *K*  *INITIAL* @@
        *KEY* D *VALUE*
        *KEY* DSNAME *VALUE* *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* SEQ *SELF* @0@
        *KEY* NOSEQ *SELF* @1@ *END*
    *PARM* *NAME* *P*  *END*
    *PARM* *NUM* *P* *INITIAL* @@ *END*
*KEYWORD* CHANGE *RTN* SUB4 *DL-EX-LIST* @¬"'=()@
*KEYWORD* CH    *RTN* SUB4 *DL-EX-LIST* @¬"'=()@
    *PARM* @NRANGE @ *INITIAL* @@ *END*
    *PARM* *STRING* *K* *P* *INITIAL* @@
        *KEY* ¬ *CALL* @STRINGC ¬@
        *KEY* ' *CALL* @STRINGC '@
        *KEY* " *CALL* @STRINGC "@  *END*
    *PARM* *STRING* *K* *P* *INITIAL* @@
        *KEY* ' *CALL* @STRINGD '@
        *KEY* " *CALL* @STRINGD "@
        *END*
    *PARM* @COLUMN @ *P* *INITIAL* @@   *END*
    *PARM* *NUM* *K* *P* *INITIAL* @-1@
        *KEY* NOTEXT *SELF* @0@
        *KEY* NOLIST *SELF* @1@
        *END*
*END-TABLE*
```

APPENDIX G -- CRBE EXAMPLE

TABLE

```
 CRBE EXAMPLE---GEORGE  07/22/70  12:50:25.960

NRANGE::'"()¬=::
:BNUM ***:-1:
:LNUM *P**:-1:
:VAL  *P**:-1:
VAL::'"()=¬::
:*NUM**K*:-1:
::(:*VALUE*:):
BNUM::()'"=¬::
:*NUM***:-1:
::FIRST:*SELF*:0:
LNUM::()¬='"::
:*NUM***:-1:
::LAST:*SELF*:-2:
ARANGE::'"()¬=::
:*STRING**K*::
::¬:*CALL*:STRINGA ¬:
::':*CALL*:STRINGA ':
::":*CALL*:STRINGA ":
STRINGA::'"()¬=::
:*STRING**K*::
::¬:*SELF*:¬:
:*STRING**K*::
::':*CALL*:STRINGB ':
::":*CALL*:STRINGB ":
STRINGB::'"()¬=::
:*STRING**K*::
::':*VALUE*:':
::":*VALUE*:":
:*STRING*P*K*::
::COL:*CALL*:CB :
:*NUM*P*K*:0:
::SEQ:*SELF*:0:
::NOSEQ:*SELF*:1:
COLUMN::/'"()=¬::
:CB  **K*::
::COL:*VALUE*::
CB::'"()=¬::
:CBB  **K*::
::=:*VALUE*::
CBB::'"(),¬=::
:*NUM**K*:-1:
::(:*VALUE SHORT*:),:
:*NUM*P*K*:-1:
::,:*VALUE*:):
::):*SELF*:-1:
DSPEC::=,().::
:NAM  **K*::
::=:*VALUE*::
:*NAME*P*K*::
::(:*VALUE*:):
:EQNAM  *P*K*::
::,:*VALUE*::
```

```
EQNAM::=()::
:EQNAMB **K*::
::VOL:*VALUE*::
::V:*VALUE*::
EQNAMB::=()::
:*NAME**K*::
::=:*VALUE*::
STRINGC::'"()¬::
:*STRING**K*::
::¬:*SELF*:¬:
:*STRING*P*K*::
::':*CALL*:STRINGD ':
::":*CALL*:STRINGD ":
STRINGD::'#()¬"::
:*STRING**K*::
::':*VALUE*:':
::":*VALUE*:":
NAM::,().::
:*NAME***::
::,:*SELF*:ACTIVE:
:NAMC **K*::
::.:*VALUE*::
NAMC::,().::
:*NAME***::
:NAMC **K*::
::.:*VALUE*::
LIST:SUB1:()¬'"=::
L:SUB1:()¬'"=::
:NRANGE ***::
:ARANGE *P**::
SAVE:SUB2:.()::
S:SUB2:.()::
:NAM ***::
:CBB (*P*K*::
::(:*VALUE*::
:*NUM*P*K*:-1:
::KEEP:*SELF*:0:
::PURGE:*SELF*:1:
:*NUM*P*K*:-1:
::REPLACE:*SELF*:0:
::REPL:*SELF*:0:
BRING:SUB3:=()::
B:SUB3:=()::
:DSPEC **K*::
::D:*VALUE*::
::DSNAME:*VALUE*::
:*NUM*P*K*:-1:
::SEQ:*SELF*:0:
::NOSEQ:*SELF*:1:
:*NAME*P**::
:*NUM*P**::
CHANGE:SUB4:¬"'=()::
CH:SUB4:¬"'=()::
:NRANGE ***::
:*STRING*P*K*::
::¬:*CALL*:STRINGC ¬:
::':*CALL*:STRINGC ':
::":*CALL*:STRINGC ":
:*STRING*P*K*::
::':*CALL*:STRINGD ':
::":*CALL*:STRINGD ":
:COLUMN *P**::
:*NUM*P*K*:-1:
::NOTEXT:*SELF*:0:
::NOLIST:*SELF*:1:
$$$
```