IMPLEMENTATION OF THE ANALYSIS PART OF A PASCAL TRANSLATOR

by

H. Cem Ataç

B.S. in C.S., Middle East Technical University, 1981

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of

Master

of

Science

Boğaziçi University

1983

IMPLEMENTATION OF THE ANALYSIS PART OF A PASCAL TRANSLATOR

APPROVED BY

Y.Doç.Dr. Oğuz TOSUN O. Tosun
(Thesis Advisor)

Y.Doç.Dr. M.Akif EYLER M Akif Eyler

Y.Doç.Dr. Selahattin KURU

DATE: 5. 8. 1983

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis supervisor Dr.Tunç Balman for all his contributions, encouragement, help and understanding during all the stages of this thesis.

I would also like to thank to other members of my graduate committee, Dr.Akif Eyler, Dr.Oğuz Tosun and Dr.Selahattin Kuru.

I would further like to thank to computer center personnel especially Mr.Zafer Erbaş, and Mr.Tamer Şıkoğlu for their valuable comments and helps in completing this thesis.

Finally, I would like to thank to my father, Naci Ataç, and my friend Bülent Ulutuna for the drawings of this thesis.

ABSTRACT

This thesis describes the implementation of the analysis part of a single pass translator for the programming language STANDARD PASCAL. The translation process includes syntactic analysis and semantic analysis at the declaration level, but excludes code generation. The details of the implementation are described, with a short description of alternative approaches in each section. The source and test runs of the program are given in the appendixes A and B respectively.

ÖZET

Bu tez, programlama dili STANDARD PASCAL için geliştirilen, tek geçişli bir çeviricinin gerçekleştirimini anlatır. Çeviri işlemi, sözdizimsel ve kısmen anlamsal çözümlemeyi içerir, ancak kod üretimi göz önüne alınmamıştır. Projenin detayları her bölümde değişik yaklaşım yöntemleri belirtilerek açıklanmıştır. Kaynak izlence ve test geçişleri, sırasıyla Ek A ve Ek B de verilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

<...> The angular brackets are used to distinguish non-terminals from terminals. The element delimited by these brackets is supposed to be a non-terminal.

$\Longrightarrow$ For a grammar G, we say that the string v *directly produces* the string w, written

$$v \Longrightarrow w$$

if we can write

$$v{=}xUy \text{ and } w = xuy$$

for some strings x and y, where U::=u is a rule of G.

$\Longrightarrow +$ For a grammar G, we say that the string v *produces* the string w, written

$$v \Longrightarrow + \; w$$

if there exists a sequence of direct derivetions

$$v \Longrightarrow u_o \Longrightarrow u_1 \text{ ------}\Longrightarrow U_n$$

where n > 0.

$\Longrightarrow *$ This symbol can be equivalently written as;

$$v = w \text{ or } v \Longrightarrow +w$$

::= Used as an abbrevation for "defined as".

I. INTRODUCTION

This thesis describes a partially completed PASCAL translator (BUPASCAL) implemented on the UNIVAC 1106 of Boğaziçi University. The program is written in standard PASCAL and occupies approximately 60 K words of memory.

BUPASCAL accepts programs written in STANDARD PASCAL as input and generates a listing of its input with the associated error messages. After each input program line is processed, error codes corresponding to the errors fetched (if any) are printed with their positions on the line, just after the line is printed. For example, errors of line 5 of the following statements are printed as;

```
line≠        source lines
  4          J := 3
  5          K := L + ;     M := 11 ;
             1        2          3

ERROR   14 ... POSITION 1
ERROR    2 ... POSITION 2
ERROR  205 ... POSITION 3

  6          if A > 10 then  C := 3 ;
```

When the input program is completely processed, explanations of the errors detected are printed as follows;

```
                 ERROR EXPLAINATIONS
      ***ERROR  14 : ';' EXPECTED
      ***ERROR   2 : IDENTIFIER EXPECTED
      ***ERROR 205 : NULL STRING NOT ALLOWED
```

Translator is a misleading term for BUPASCAL since it is actually an analyzer and it does not generate code. In addition to error detection and reporting, BUPASCAL collects information about the identifiers of the input program, into a symbol table. This table is optionally dumped as the source listing is produced.

The major components of BUPASCAL are

(1) Lexical analyzer -- described in section II
(2) Syntactic analyzer -- described in section III
(3) Error recovery -- described in section IV
(4) Symbol table -- described in section V

II. LEXICAL ANALYSIS

A. Definition

The lexical analyzer is a part of a translator which reads the source program one character at a time and constructs the source program symbols (identifiers, keywords, constants and delimeters).

One may justly ask, why the lexical analysis can not be incorporated into the syntax analysis. After all, we can use BNF to describe the syntax of symbols. For example, PASCAL identifiers can be described by

```
<identifier>  ::= <letter> {  <letter> | <digit>}o
<letter>      ::= A|B|C|...  |Z
<digit>       ::= 0|1|  ...  |9
```

There are several reasons for splitting the analysis of the source program into two phases, lexical analysis and syntactic analysis;

1. Main purpose is to simplify the overall design of the translator. Considering a large portion of compile-time is spent in scanning characters, by seperation we can concentrate solely on reducing this time.
2. The syntax of symbols can be described by very simple grammars. If we seperate scanning from syntax recognation we can develop efficient parsing techniques which are particularly well suited for these grammars.

3. Generally speaking, syntax analyzer requires much more programming effort than the lexical analyzer does. Our aim must be, therefore to ease life for the syntax analyzer. So, some cumbersome functions such as keeping track of line numbers, producing an output listing, ignoring blanks and comments etc. can be performed by the lexical analyzer.
4. Since the lexical analyzer returns a symbol instead of a character, the syntax analyzer actually gets more information about what to do at each step. Moreover, the steps required for the syntax checking will be decreased.
5. Seperation allows us to write one syntactic analyzer and several lexical analyzers (which are simpler and easier to write) one for each source program representation and/or input device. Each lexical analyzer translates the symbols into the same internal form used by the syntactic analyzer.

A lexical analyzer may be programmed as a seperate pass which performs a complete lexical analysis of the source program and which gives to the lexical analyzer a table containing the source program in an internal symbol form. Alternatively, it can be a subroutine called by the syntax analyzer, whenever the syntax analyzer needs a new symbol (Figure 2.1).

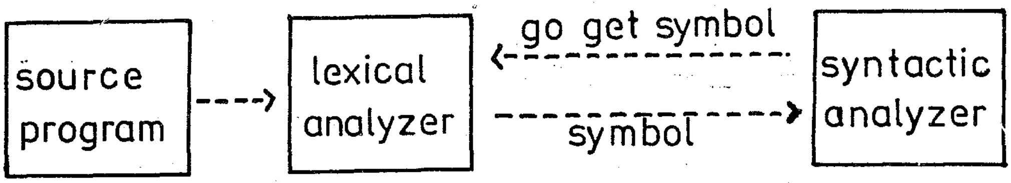


Figure 2.1- Lexical Analyzer as a Subroutine of Syntactic Analyzer.

When called lexical analyzer recognizes the next program symbol and passes it to the syntactic analyzer. This alternative is generally better, because the whole internal source program need not to be constructed and kept in memory.

In this thesis, lexical analyzer is implemented in this manner. It is a PASCAL procedure named SCAN. Implementation of SCAN will be explained in detail within the rest of this section.

B. The Output Of a Lexical Analyzer

A lexical analyzer builds an internal representation of each symbol. In most cases this is a fixed length integer. As opposed to variable length strings, which are the actual symbols, these internal representations are much easier to manipulate and parse.

We include in this internal representation a constant meaning "identifier" and another for "integer" etc. That is, all identifiers have the same internal number to represent them. This is natural, because the term "identifier" is a terminal symbol to the syntax analyzer, and which identifier it happens to be is of no consequence. However, the identifier itself is needed by the symbol table management routines, so it must be stored somewhere. Similarly, it is enough for the syntax analyzer to know an integer is met (whatever its value is), but the semantic analyzer should know its internal value.

The solution is to output two values; the first is the internal representation and the second is the actual symbol itself or a pointer to it. Thus, the output of the lexical analyzer will consist of two fields.

1. CLASS : This is the internal representation of the symbol scanned. This field is utilized by the syntax analyzer.

2. VALUE : This field gives further and more precise information pertaining to the class of the recognized symbol. Only a few classes of symbols make use of this field and it is utilized by the semantic analyzer.

The information placed in the VALUE field is dependent on the CLASS field and can sometimes be empty. VALUE field may be used in two different ways. First use is, in the case of identifiers or literal constants to keep actual symbol or internal values as explained above.

Second, in some cases further information is required to indicate which particular instance of a class, the syntactic primitive is. An example of this is the PASCAL <multiplication operation> which has five possible instances, -,*,/,div,mod, and. Although, the precise mature of the multiplication operation does not affect the syntax analyzer, it has to be known for the semantic analyzer.

C. State Transition Diagrams

Before discussing the implementation of SCAN, I would like to mention about state transition diagrams, which I believe, is the best way, to show how a symbol is to be parsed. Figure 2.2 shows a simple state diagram.

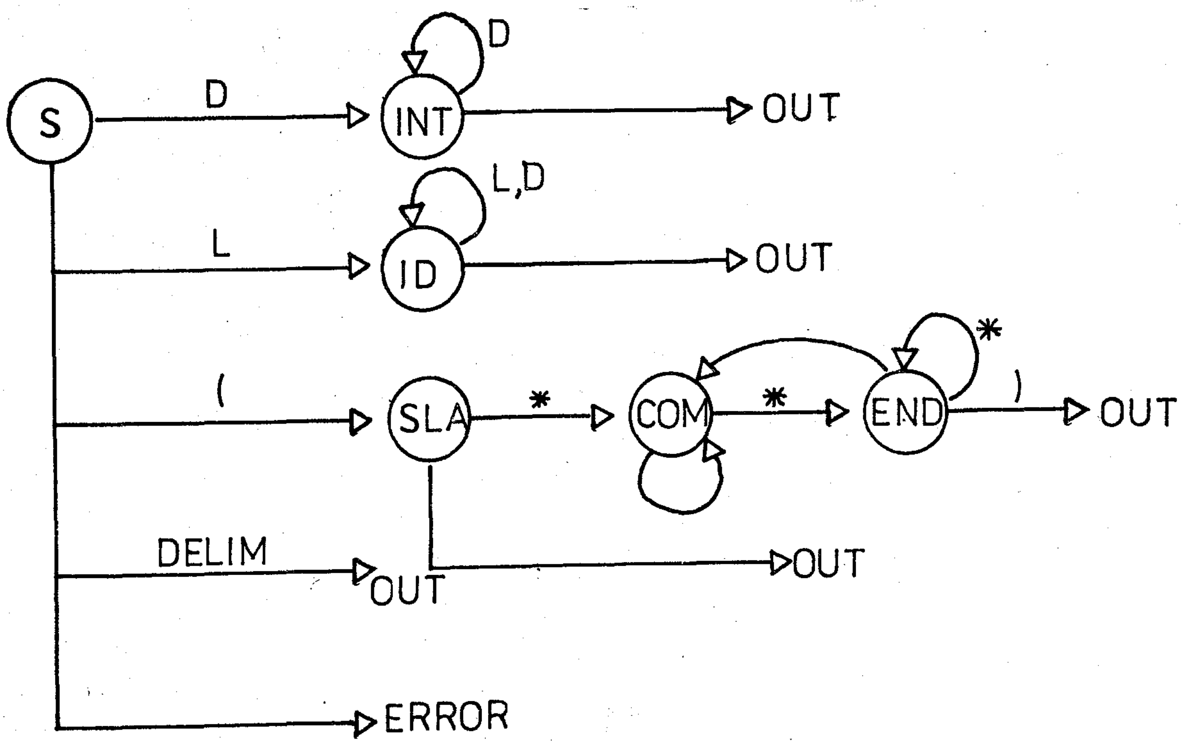


Figure 2.2- State Diagrams Scanning Symbols.

In this figure, the label D is an abbrevation for the labels 0,1,2,...,9. That is, D represents the class of digits. This is done to keep the diagram simple. Similarly, the label L represents the class of letters A,B,C,...,Z, and DELIM represents the class of single character delimeters +,-,*,/ etc. Note that, character "(" is not in this class, since it must be handled in a special manner.

Several of the arcs have no label on them. These are the arcs to be taken, if the character scanned is one that does not explicitly appear on an arc. For example, when in state INT, as long as we scan a digit we stay in state INT. If a non-digit character is scanned we proceed along the arc to OUT.

The arc OUT indicates the class and value of the symbol recognized. That is, by arc OUT we mean that we have detected the end of a symbol and we want to leave the lexical analyzer.

Having provided a transition diagram, there are two approaches, to program the lexical analyzer. First is, to represent transition diagram in core as a set of tables indexed by states and characters and then move within these tables. Second approach, is to develop the lexical analyzer algorithmically. Here we do not have any explicit tables. A transition diagram is viewed as a specialized kind of flowchart. States of a transition diagram correspond to the boxes of a flowchart. So, according to the arrows connecting these states, program structures (loops, if-then-else etc.) are designed until an out arc is reached. This approach is particularlу useful when the program is a lexical analyzer, because the action taken is highly dependent on what characters have been seen recently.

The second approach is chosen (except for identifiers for which a set of transition tables is used) in the implementation of SCAN. SCAN uses a case statement based on the first character of the incoming symbol, to determine which arc to take from the initial state S. Following characters are handled as described above.

When an ERROR arc is reached SCAN takes the corrective action and tries again. Errors detected by SCAN is referred as LEXICAL TIME errors and they will be presented under the heading E.

D. Implementation

Within this section, the PASCAL procedure SCAN, which is the lexical analyzer of BUPASCAL will be explained.

1. Symbols of PASCAL

Symbols of PASCAL can be classified into five groups. In each case the class field represents the type of syntactic primitive, and the value field, if utilized, supplies further information.

i) IDENTIFIERS

Identifiers can be predefined or user defined. Predefined identifiers are

- Subprogram identifiers such as READ, WRITE, EOLN etc.
- Type identifiers such as INTEGER, REAL, BOELEAN etc.
- Const identifiers such as TRUE, FALSE, MAXINT etc.

In general all identifiers (constants, variables, procedures, functions, types, fields etc) whether user defined or predefined are identified by the same class code. The only difference between predefined and user defined identifiers is that, predefined identifiers are located during initialization into the symbol table. But this is not a concern of SCAN.

Value field is unutilized. Instead a global string variable keeps the actual symbol, for symbol table search.

ii) KEYWORDS

These are the syntactic primitives that serve to identify the different types of PASCAL statements and program components.

They have the same syntax with identifiers, but SCAN seperates them from identifiers, by a set of transition tables, thus eliminating a need for a search of a table of keywords. There are thirty keywords and each keyword is represented by a distinc class code. Note that, value field is again unutilized.

iii) LITERAL CONSTANTS

In this category we include unsigned integers, unsigned reals, character constants and string constants. Thus, four internal codes are required to indicate the type of the constant in the class field. The value field points to the location that contains the value of the constant in the dynamic constant table.

iv) OPERATORS

Pascal operators can be classified into three groups, each group requiring a distinct class code.

1. Relational operators
   These are "<", ">", "=", "<>", ">=", "<=" and in. The value field indicates the precise operator within the group.

2. Adding operators
   These are "+", "-" and or and the value field pinpoints the precise operation.

3. Multiplying operators
   Similar to the previous two groups except the operators are "*", "/", div, mode, and.

Note here that the generalizations in groups (1) and (2) are not accurate. The symbol "=" when used in type and const statements is not a relational operator. Similary "+" and "-" can sometimes be used as signs.

This discrepancy is avoided by taking corrective action during semantics check.

v) DELIMETERS

In this category we include all other single character e.g. ".", ";", "(", " :") and double character delimeters (e.g. "..", ":=") of PASCAL.

Each of these delimeters will be represented by a unique class code and the value field will not be utilized.

SCAN is also responsible to detect and ignore seperators which are principally for the benefit of the human reader. Seperators are blanks, end-of-lines or comments. They have no effect on the meaning of the program.

2. Data Structures to Represent PASCAL Symbols

At this point, we can present data structures used in recognation of PASCAL symbols. These structures are declared globally (Table 2.1).

In these declarations, CLASS and VALUE fields are gathered within a record type named TOKENS. The record contains three variants which states different uses of VALUE field.

TABLE 2.1- Data Structures for the Lexical Analyzer.

```
TYPE
   SYMBOL = (* ...SYMBOL CLASSES...   *)
          (        IDENT,      INTCONST,     REALCONST,   CHARCONST,
             STRINGCONST,        EOFPGM,         MULOP,       ADDOP,
                   RELOP,       LPARENT,       RPARENT,    LBRACKET,
                RBRACKET,         COMMA,     SEMICOLON,      PERIOD,
                  DOTDOT,         ARROW,         COLON,      ASGNOP,
                 ARRAYSY,       BEGINSY,        CASESY,     CONSTSY,
                    DOSY,      DOWNTOSY,        ELSESY,       ENDSY,
                  FILESY,         FORSY,     FORWARDSY, FUNCTIONSY,
                  GOTOSY,          IFSY,       LABELSY,       NOTSY,
                    OFSY,      PACKEDSY,        PROCSY,   PROGRAMSY,
                RECORDSY,      REPEATSY,         SETSY,      THENSY,
                    TOSY,        TYPESY,       UNTILSY,       VARSY,
                 WHILESY,        WITHSY,       OTHERSY         ) ;

   OPERATOR =
          (         ASTR,          RDIV,          IDIV,       ANDOP,
                    IMOD,          PLUS,         MINUS,        OROP,
                    LTOP,          LEOP,          GEOP,        GTOP,
                    NEOP,          EQOP,          INOP         ) ;

   LITERALS      =  INTCONST .. STRINGCONST       ;
   OPERATORS     =  MULOP    .. RELOP             ;
   DELIMETERS    =  LPARENT  .. ASGNOP            ;
   KEYWORDS      =  ARRAYSY  .. WITHSY            ;

   MULTIPLICATION_OPERATORS =  ASTR .. IMOD       ;
   ADDITIONAL_OPERATORS     =  PLUS .. OROP       ;
   RELATIONAL_OPERATORS,    =  LTOP .. INOP       ;

   CSTCLASS    = (INTGR, REEL, STRING, KHAR)                  ;
   STRGRANGE   =     0 ..  78                                 ;
   CHARSTRING  = PACKED ARRAY [STRGRANGE] OF CHAR             ;

   CSTADDR     = @ CONSTANT ;
   CONSTANT    = PACKED RECORD
                   CASE CLASS : CSTCLASS OF
                      INTGR : (IVAL : INTEGER            )    ;
                      REEL  : (RVAL : REAL               )    ;
                      KHAR  : (ORDCH: INTEGER            )    ;
                      STRING: (SLGTH: STRGRANGE               ;
                               SVAL : CHARSTRING         )
                END ;

(* ..FOLLOWING ENUMERATED TYPE INDICATES HOW THE VALUE
     FIELD WILL BE UTILIZED.                               *)

   TKNCLASS    = (NOTNEEDED, OPRTR, DRADRES)                  ;

(* ..OUTPUT TYPE OF SCAN IS THE FOLLOWING RECORD WHERE
     VARIANTS OP,CSTADR DENOTES DIFFERENT USES OF VALUE
     FIELD..                                               *)

   TOKENS      = RECORD
                   CLASS : SYMBOL   ;
                   CASE TKNCLASS OF
                      OPRTR       : ( OP     : OPERATOR )    ;
                      DRADRES     : ( CSTADR : CSTADDR  )    ;
                      NOTNEEDED   : ( )
               END ;


VAR     TOKEN : TOKENS ;
```

3. Algorithm of the Lexical Analyzer

SCAN uses the following variables and routines.

1. CH : CH is a global variable which will always hold the current character of the source program being scanned.
2. IDENTIFIER: is a location which will contain the string of characters making up the symbol.
3. NEXTCHAR : is a function to return the next source program character. NEXTCHAR will take care of reading and printing next source line, when it detects end of line, while trying to scan next source character.
4. READLINE : is a procedure which reads the next source program line into an internal buffer. Function NEXTCHAR actually uses this buffer. READLINE is called whenever the buffer is wholly utilized.
5. PRINTLINE : is a procedure which dumps the above mentioned buffer. It also prints the errors of the current line (buffer) if there are any.

Figure 2.3 gives the transition diagram of whole PASCAL symbols.

SCAN uses the following data structure to implement arcs for single character delimeters.

<u>var</u> CHRCLASS : <u>array</u> (.CHAR.) <u>of</u> TOKENS;

It will be initialized as

```
CHRCLASS ['['].CLASS    := LBRACKET ;
CHRCLASS ['@'].CLASS    := ARROW    ;
CHRCLASS [')'].CLASS    := RPARENT  ;
CHRCLASS [']'].CLASS    := RBRACKET ;
CHRCLASS [','].CLASS    := COMMA    ;
CHRCLASS [':'].CLASS    := COLON    ;
CHRCLASS [';'].CLASS    := SEMICOLON;

WITH CHRCLASS ['+'] DO BEGIN CLASS:= ADDOP ; OP:= PLUS   END ;
WITH CHRCLASS ['-'] DO BEGIN CLASS:= ADDOP ; OP:= MINUS  END ;
WITH CHRCLASS ['/'] DO BEGIN CLASS:= MULOP ; OP:= RDIV   END ;
WITH CHRCLASS ['*'] DO BEGIN CLASS:= MULOP ; OP:= ASTR   END ;
WITH CHRCLASS ['='] DO BEGIN CLASS:= RELOP ; OP:= EQOP   END ;
```

TABLE 2.2- Algorithm of the Lexical Analyzer

```
PROCEDURE SCAN ;
    LABEL    1 ;
    PROCEDURE PRINTLINE ; BEGIN ... END ;
    PROCEDURE READLINE  ;
    BEGIN  IF EOF(INPUT) THEN
       TOKEN.CLASS := EOFPGM ; (* SO RETURN FROM SCAN *)
    END ;
    FUNCTION NEXTCHAR : CHAR ; BEGIN ... END ;


BEGIN   (* SCAN *)
 1  :  WHILE CH=' ' DO
       CH := NEXTCHAR ; (* IGNORE BLANKS *)

    CASE CH OF

       'A'..'Z'   : .... ;   (* IDENTIFIERS                   *)
       '0'..'9'   : .... ;   (* NUMERIC CONSTANTS             *)
       '<'        : .... ;   (* FOR THESE DELIMETERS FOLLOW   *)
       '>'        : .... ;   (* THE ACTIONS OF CORRESPONDING  *)
       ':'        : .... ;   (* ARCS SPECIFIED IN PASCAL      *)
       '.'        : .... ;   (* TRANSITION DIAGRAM.           *)

       '('        : (* THIS CASE LABEL IS PERFORMED TO SHO
                         HOW A LABEL WILL BE TREATED IN GENERAL *)

             BEGIN
                CH := NEXTCHAR ;

                IF CH = '*' THEN (* COMMENTS *)
               BEGIN

                   REPEAT
                      REPEAT UNTIL NEXTCHAR = '*'
                   UNTIL NEXTCHAR = ')' ;

                   CH := NEXTCHAR ; GOTO 1

                END

                ELSE IF CH ='.' THEN
                   BEGIN
                      TOKEN.CLASS := LBRACKET ;
                      CH   := NEXTCHAR
                  END

                ELSE TOKEN.CLASS := LPARENT

             END ;    (* CASE LABEL '(' *)

       OTHERWISE

             BEGIN
                TOKEN.CLASS := CHRCLASS [CH] ;

                IF TOKEN.CLASS = OTHERSY THEN
                   BEGIN
                       ERROR (6); (* ILLEGAL SYMBOL *)
                       CH := NEXTCHAR ;
                       GOTO 1
                    END
             END

    END          (*   CASE   *)
END ;            (*   SCAN   *)
```

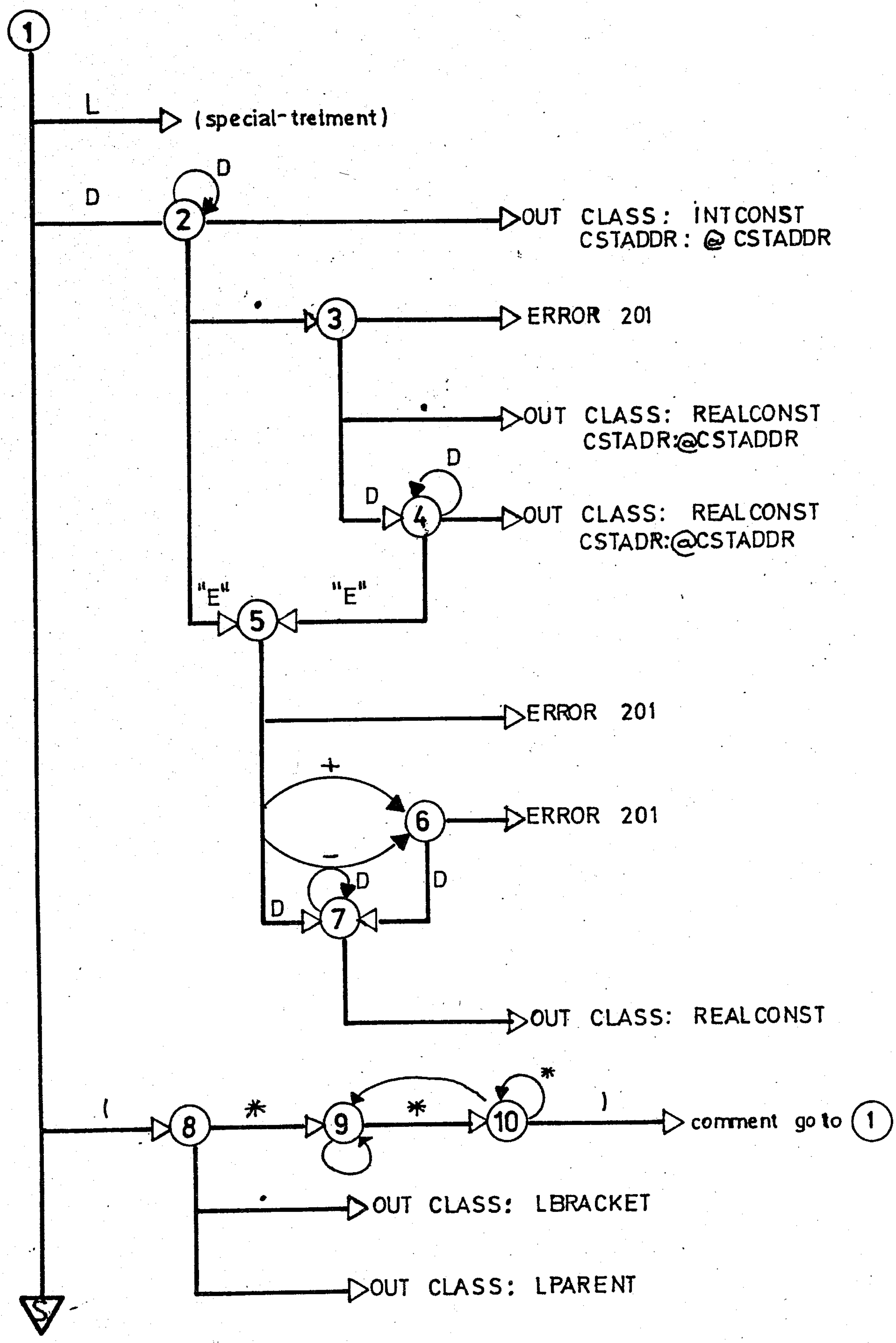


Figure 2.3- State Diagram for PASCAL.

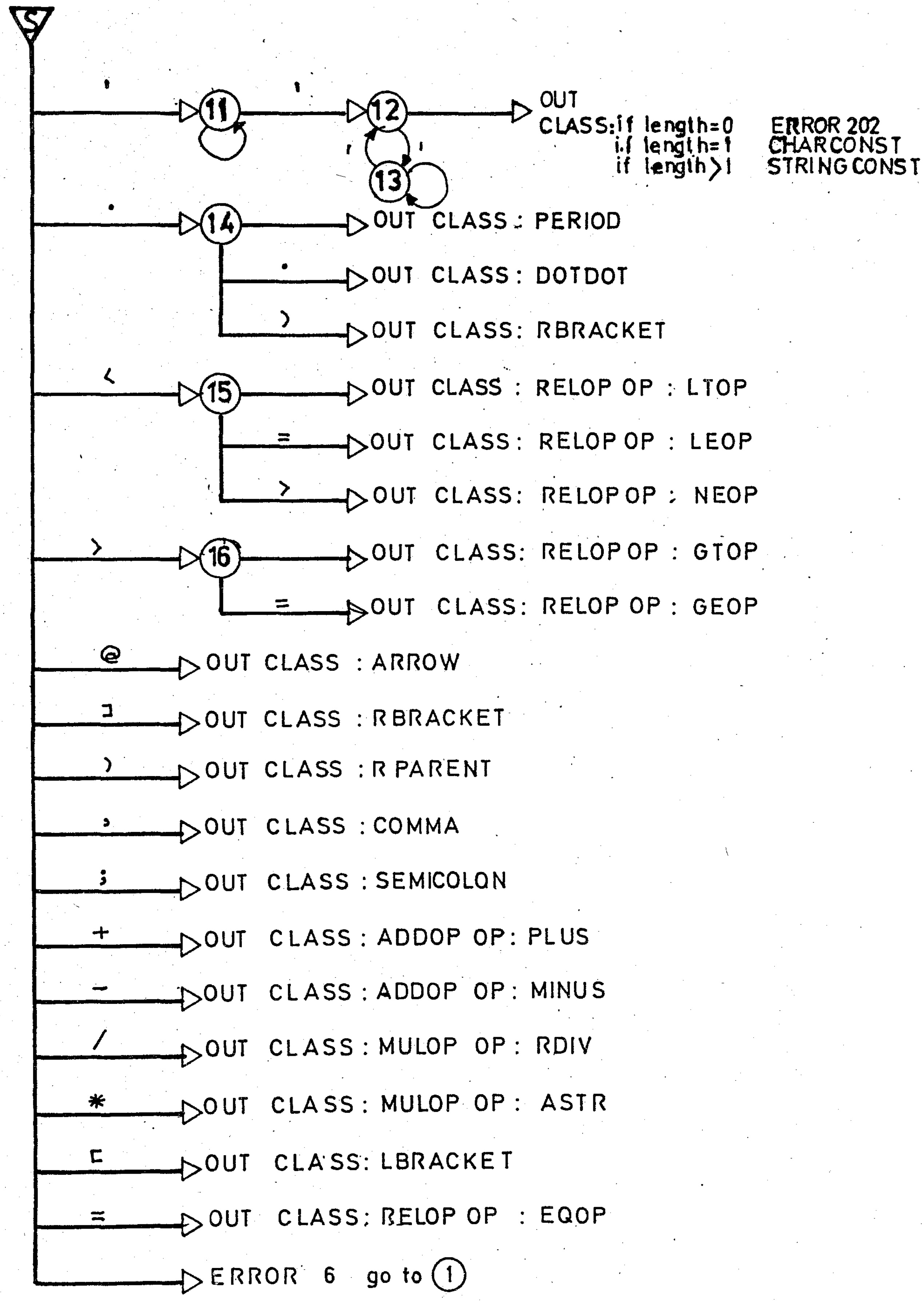

S
'
11
'
12
'
'
13
OUT
CLASS:if length=0 ERROR 202
if length=1 CHARCONST
if length>1 STRINGCONST
.
14
OUT CLASS : PERIOD
.
OUT CLASS: DOTDOT
)
OUT CLASS: RBRACKET
<
15
OUT CLASS : RELOP OP : LTOP
=
OUT CLASS: RELOP OP : LEOP
>
OUT CLASS: RELOPOP : NEOP
>
16
OUT CLASS: RELOPOP : GTOP
=
OUT CLASS: RELOP OP : GEOP
@
OUT CLASS : ARROW
]
OUT CLASS : RBRACKET
)
OUT CLASS : R PARENT
,
OUT CLASS : COMMA
;
OUT CLASS : SEMICOLON
+
OUT CLASS : ADDOP OP: PLUS
-
OUT CLASS : ADDOP OP : MINUS
/
OUT CLASS : MULOP OP : RDIV
*
OUT CLASS : MULOP OP : ASTR
[
OUT CLASS: LBRACKET
=
OUT CLASS: RELOP OP : EQOP
ERROR 6 go to 1

Figure 2.3- Continued.

All other characters will have a CLASS field of value OTHERSY. These are characters that have not been included in PASCAL character set, so if met an error message should be given.

Also note that a class code, EOFPGM is included within the user defined scalar type SYMBOL, which indicates there are no more symbols to the syntactic analyzer.

The procedure SCAN has the algorithm of table 2.2 (Assume result is returned in the global variable TOKEN).

Treatment of case labels in algorithm of table 2.2 are easy and straightforward. That is, once starting character of a symbol is found, it is the task of the corresponding case label to complete the syntax of the symbol. (Note that, SCAN assures that when it returns, the next character will always be scanned).

Only the case label related to identifiers, is specially treated. Identifiers and keywords both have the same syntax. So, the keywords of the language could be initially classified as identifiers and the correct symbol asserted after consulting a table of keywords. Due to the additional table search, this method is slower than the direct recognition of keywords through state transition tables.

Thus assuming we have two keywords ENDE, ELSE the arc labelled L in figure 2.3 will be modified to include keywords ENDE, ELSE (figure 2.4. Note that keyword ENDE used in this figure is not a PASCAL keyword).

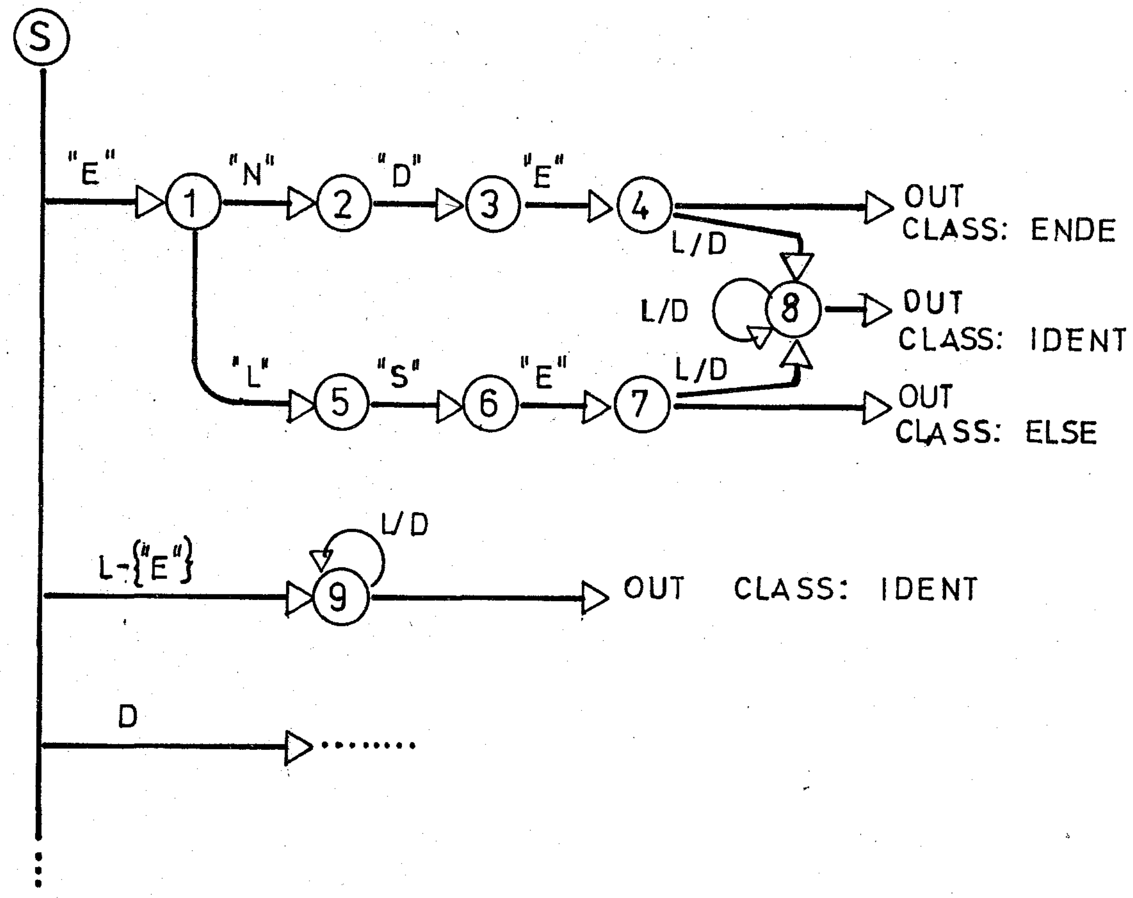


Figure 2.4- State Diagram to Recognize Keywords.

In figure 2.4 symbols L,D represent the set of letters and digits respectively as in fig. 2.3. The label L/D means letter or digit and L-{"E"} denotes the set of letters excluding the character "E". This diagram actually contains all PASCAL keywords but for simplicity I assumed only the existence of two keywords.

A set of tables is used to represent this transition diagram. The set consists of four arrays indexed by state numbers. These arrays are declared as;

```
var  DEFAULT       : array [0..171]  of TOKENS;
     NEXT, CHECK   : array [0..168]  of INTEGER;
     BASE          : array [0..171]  of INTEGER;
```

BASE array is used to determine the base location of the entries for each state stored in NEXT and CHECK arrays.

DEFAULT array indicates the class of symbol fetched.

Assuming we are at case label 'A'..'Z' in the procedure SCAN, (thus CH contains a letter initially) table 2.3 gives the algorithm used (BAZ and S are variables declared of type integer).

To compute the transition for state S on input character CH, the pair of arrays NEXT and CHECK is first consulted. In particular, the algorithm finds their entries for state S in location BAZ := BASE [S] + ORD(CH). NEXT [BAZ] is taken to be the next state for S on input CH if CHECK [BAZ] = S. This procedure is repeated within a loop until CHECK [BAZ] <> S for some state S.

When we exit from the loop, the DEFAULT entry corresponding to the last state S, shows whether an identifier or a keyword (if so which one) is met.

For example, suppose state 1 in figure 2.4 is indexed 5 in numbering states (that is S = 5 initially). Then a value in chosen for BASE [5] and value 5 is intered into CHECK [BASE [5] + ORD ('N')] and CHECK [BASE[6] + ORD ('L')]. The next states on 'N' and 'L' for 5, are entered into the corresponding entries of the NEXT array.

TABLE 2.3- Algorithm to Compute Transitions

```
S   := ORD(CH) ;   (* INITIAL STATE FOR BASE ARRAY *)
CH  := NEXTCHAR ;
BAZ := BASE [S] + ORD (CH) ;

WHILE S = CHECK [BAZ] DO BEGIN
  S   := NEXT [BAZ] ;
  CH  := NEXTCHAR ;
  BAZ := BASE [S] + ORD (CH)
END ;    (* WHILE *)

TOKEN  := DEFAULT [S] ;
IF CH IN [ 'A'..'Z' , '0'..'9' ] THEN
  TOKEN.CLASS := IDENT ;
       (*  ... RETURN ...  *)
```

DEFAULT [5]. CLASS will be IDENT. The algorithm of Table 2.3 will make the right transitions on "N" and "L", otherwise loop will be terminated. Therefore, if input CH is any character, but "N" or "L", we shall not find CHECK [BASE[5] + ORD(CH)] = 5 and exit from the loop and token will be DEFAULT 5 , which is an IDENT.

Only, the DEFAULT entries corresponding to states 4 and 7 in figure 2.4 will contain classes representing keywords ENDE and ELSE respectively.

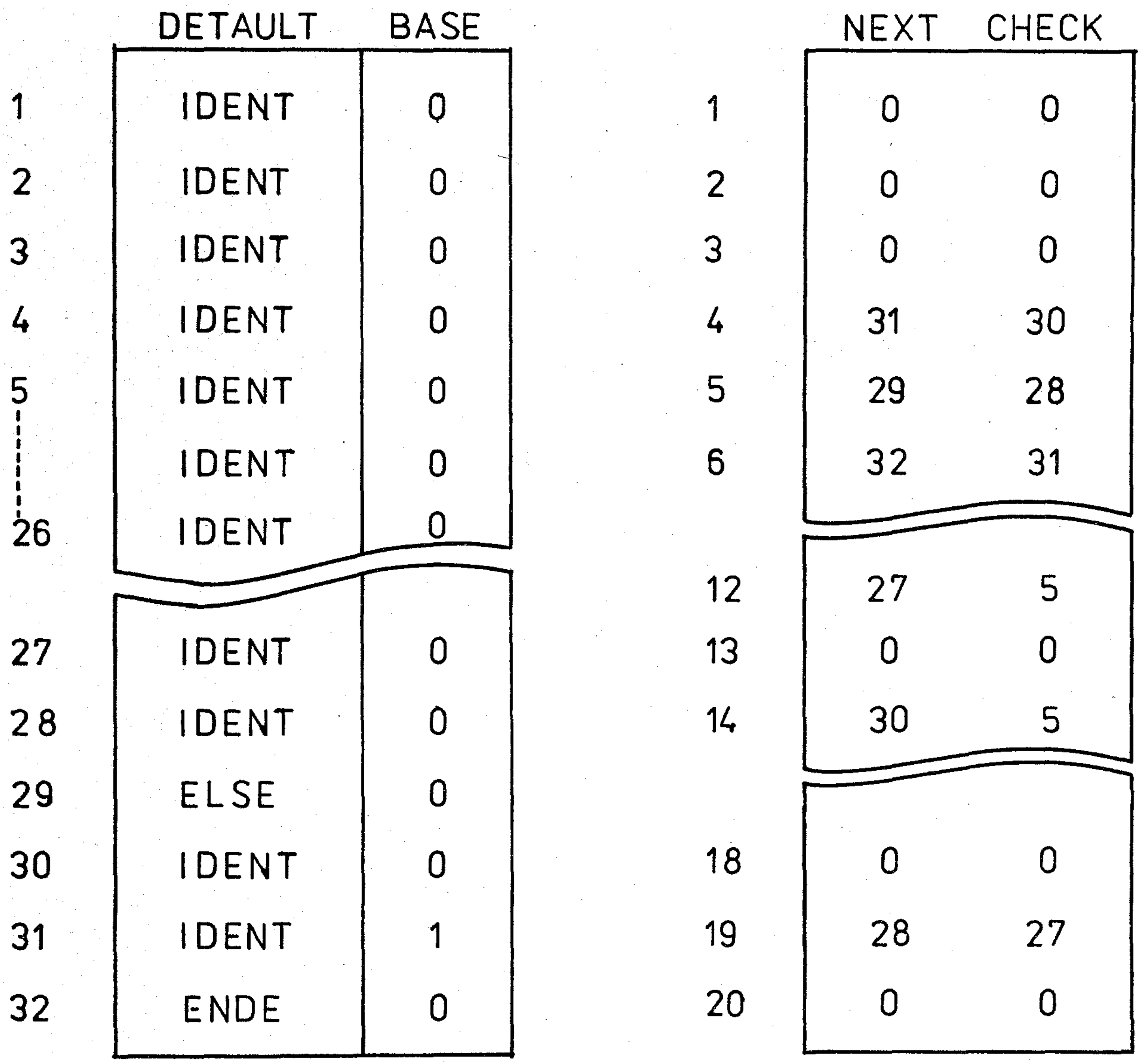
BASE values are initialized so that BAZ values in table 2.3 for different S values, do not conflict with the existing CHECK entries. Sizes of NEXT and CHECK arrays are highly dependent on these BASE values, so that they must be chosen carefully.

Assume that the input string is ELSEA. According to the algorithm we exit from the loop with DEFAULT class for the keyword ELSE. The last if statement in table 2.3 used to correct such errors. That is, as soon as we exit from the loop the last character scanned is tested. If it is a letter or digit, that means we have met an identifier, not a keyword.

Algorithm of table 2.3 is also used in recognition of operators such as mod, and, or, div etc. in addition to keywords and identifiers.

The transition tables for the diagram in figure 2.4 is shown in figure 2.5. Here it is assumed that ordinal values of letters range from 1 to 26, in alphabetical order. That is ORD ('A') = 1, ORD('B') = 2,...,ORD('Z') = 26, and so on.

Base values are all zero except for entry 31. This is because when S is 31, letter "E" is expected to recognize the keyword ENDE and if BASE 31 were 0, baz value (BASE[31] + ORD ('E')) would correspond to CHECK entry 5, which have already been utilized.



	DETAULT	BASE
1	IDENT	0
2	IDENT	0
3	IDENT	0
4	IDENT	0
5	IDENT	0
	IDENT	0
26	IDENT	0
27	IDENT	0
28	IDENT	0
29	ELSE	0
30	IDENT	0
31	IDENT	1
32	ENDE	0

	NEXT	CHECK
1	0	0
2	0	0
3	0	0
4	31	30
5	29	28
6	32	31
12	27	5
13	0	0
14	30	5
18	0	0
19	28	27
20	0	0

Figure 2.5- Transition Tables to Recognize Keywords.

E. Lexical Time Errors

SCAN is responsible of detection and reporting lexical time errors. Lexical time errors of PASCAL are defined as follows;

ERROR 6 : Illegal symbol
ERROR 201 : error in real const; digit expected
ERROR 202 : integer constant exceeds range
ERROR 205 : null string not allowed
ERROR 206 : integer part of real constant exceeds range.

The detection points of ERRORS 6,201,205 are shown in the transition diagram of fig. 3.3. Errors 202 and 206 can be detected during conversion of numbers from character code to binary form.

F. Conclusion

SCAN is designed as a hand-written lexical analyzer. For this algorithm GRIES (2) can be referred. The set of transition tables used for identifiers and keywcrds are explained in AHO-ULLMAN(1).

SCAN can work independent of the machine it is run with simple modifications. It is designed to work in FIELDATA character set, because the transition tables used for the recognition of identifiers and keywords are initialized using FIELDATA character codes. But the algorithm can be made to work in any character set, by addition of a single array.

Conversion algorithms from character code to binary form (in case of numbers) are general but errors 202 and 206 depend on the word size of the machine.

III. SYNTACTIC ANALYSIS

A. Definition

A syntactic analyzer (or PARSER) for a grammar G is a program that takes as input a string W and produces as output either a parse tree for it, if W is a sentence of G, or an error message indicating W is not a sentence of G. In case of translators, W corresponds to programs of a programming language whose grammar is defined by G.

Parsing algorithms can be classified into two categories bottom-up and top-down. The terms refer to the way in which parse trees are built. A bottom-up parser builds parse trees from the bottom (terminal nodes) to the top (root node). A top-down parser builds parse trees starting from the root node and works down to terminal nodes. In both cases the input to the parser is being scanned from left to right, one symbol at a time.

B. Bottom-Up Parsing

The bottom-up technique is to start at the string itself and try to reduce it to the distinguished symbol. Consider the sentence 35 of the following grammar for integers.

```
<N> ::= <D>|<N><D>
<D> ::= 0|1|2| ... |9
```

The first step is to reduce the 3 to <D>, yielding the sentential form <D>5. Thus the direct derivation <D>5=>35 is constructed as shown in figure 3.1a. Next step is to reduce. <D> to <N> (figure 3.1b). This proceeds until the last tree (figure 3.1d) has been formed. Note that, in such a parse, at each step a handle (leftmost simple phrase) of the sentential form is reduced.

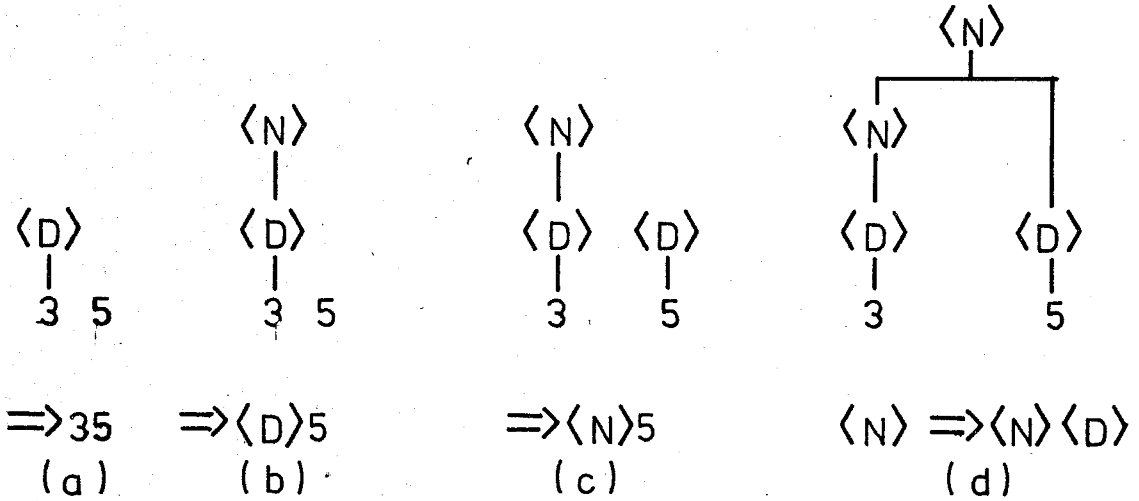


Figure 3.1- Bottom-Up Parse and Derivation of its Constructs.

Bottom-up parsers usually use a stack for reduction process. Symbols are pushed on to the stack until the right side of a production appears on top of the stack. The right side may then be replaced by (reduced to) the symbol on the left side of the production, and the process is repeated.

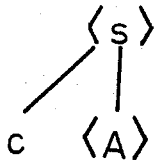
C. Top Down Parsing

Top down parsing can be viewed as an attempt to find a left most derivation for an input string. For example, consider the grammar,

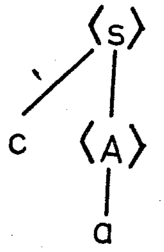
1) <S> ::= c<A>d

2) <A> ::= b|a

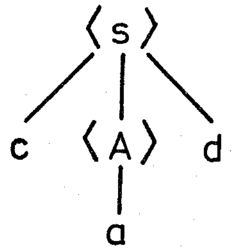
and let input be W = cad. We initially start with the root node <S>. First symbol of W matches the first symbol (which is the terminal c of rule 1. So we advance to the next input symbol, a and try to match it with the successor of terminal c in rule 1. (which is <A>). Since a non-terminal is reached, we must first expand it before proceeding. Upto here we have constructed the following partial tree.



<A> has two alternatives, among which second one matches the input symbol, that is terminal a of rule 2. Thus <A> is expanded using the second alternate to obtain the partial tree



We now consider d, third input symbol, and successor of non terminal <A> of rule 1 (since its expansion has been completed) which is the terminal d, and this matches with the input symbol.



Since we have now produced a complete parse tree for W, we halt and announce succesful completion of parsing.

Parsing algorithm used in this thesis, is a top-down technique. The algorithm is implemented as a PASCAL procedure named PARSE and it will be explained within the rest of this section.

D. Implementation

This section describes the parsing algorithm, and representation of the grammar. It also describes the problems of TOP-DOWN parsing and how they are solved by PARSE.

1. Graphic Notation of Grammars

The algorithm is centered upon the relationship between an element and its successors, and the element and its alternates. To show these relationships schematically, a graphic notation is introduced. Consider the following simple rule;

<left hand side> ::= <p><q>|<r><s>

This rule can be represented as in figure 3.2. Arcs labelled a show afternates, and arcs labelled s show successors. Finally arc labelled d means "defined as".

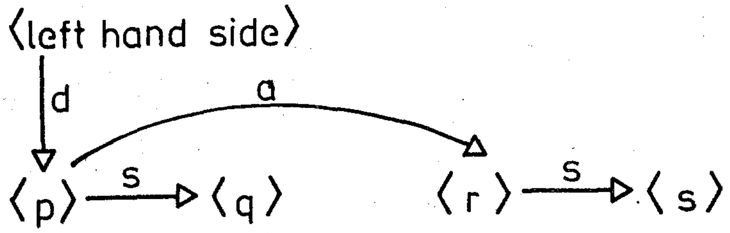


Figure 3.2- Graphic Notation of a Rule.

Tables used by PARSE to represent the grammar in core, use such alternate and successor links. So the graphic notation helps us in designing those tables.

If there is no alternate for an element, we indicate this by the absence of an alternate arc in the graph. While representing in core, alternate link of such an element will contain a dummy constant denoted by the <u>constant identifier FAIL</u>. Trying to access an alternate link whose value is FAIL causes an error. Similary to indicate end of successors, we use another dummy constant, denoted by the constant identifier OK.

2. Top Down Problems and Their Solutions

a) Direct left recursion

Assume we have a rule like <x> ::= <x> .... Then our first action to expand <x> would be to expand <x>. And next action would also be to expand <x>, since <x> is always the first rule of the expansion of <x>. Thus we would be in a deadlock creating a loop around <x>.

Best way to get rid of direct left recursion is to write the rules using iterative notation. Consider the following rule

<E> ::= <E>+<T>|<T>

It can be written as;

$<E> ::= <T>\{+<T>\}_N$

where N corresponds to the minumum number of iterations which could be zero. Iteration can be represented in successor alternate graphs easily. In this case, the last element in a { } pair points to the first element in the { } pair, as its successor. And the first element's alternate link will contain OK value. OK value used in an alternate link shows optionality and means end of the expansion of the current non-terminal just as an OK value met in a successor link.

Figure 3.3 shows the graphical representation of the rule $<E> ::= <T>\{+<T>\}_N$, when N is 0.

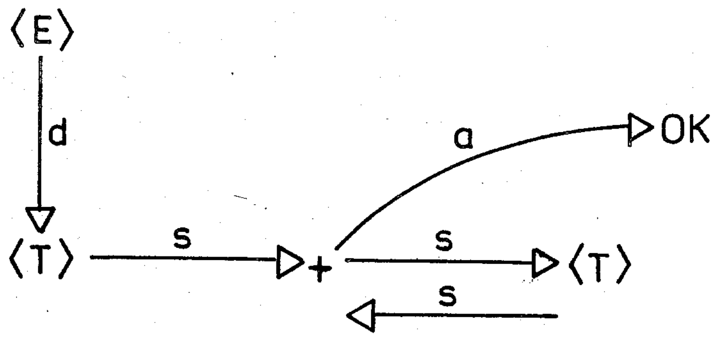


Figure 3.3- Graph of an Iterated Rule (N=0).

If N is greater than 0, the graph gets larger But N is usually 0, and there are very few rules requiring N greater than 0. Figure 3.4 shows the same graph when N is 1.

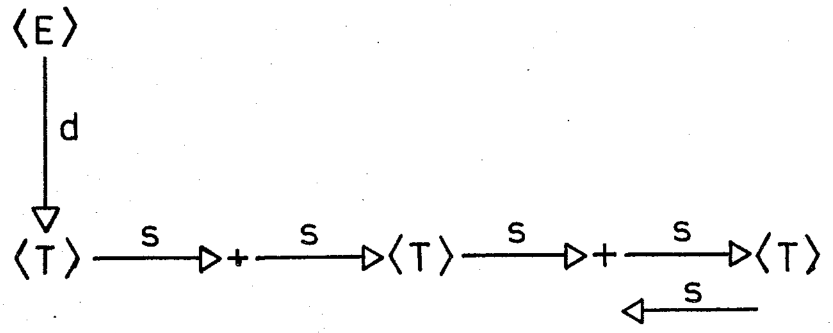


Figure 3.4- Graph of an Iterated Rule (N=1).

b) General left recursion

Assume we have rules like <U> ::= <V>x and <V> ::=<U>y. These rules yield <U> ::= <U>yx. There is no simple way of getting rid of such rules, except for manually checking.

c) Backups

Backup conditions rise when we are expanding non-terminals. That is, when we met a non-terminal we first go to its definition, and try to parse it. And if it S definition also contains non-terminals in it S definition we proceed similarly until success has been reached at some level. If during this process, a failure occurs at any level, we must back-up to a higher level, to try the alternate of the non-terminal that caused the failure. Backup process may continue upto the initial level where we first met a non-terminal.

Backups should be avoided because;

1) It is time consuming and inefficient
2) If semantics are being performed as each syntactic element is identified, then they have to be undone
3) If code generation is being done, the code generated needs to be erased upto a point
4) Backups make error recovary very hard.

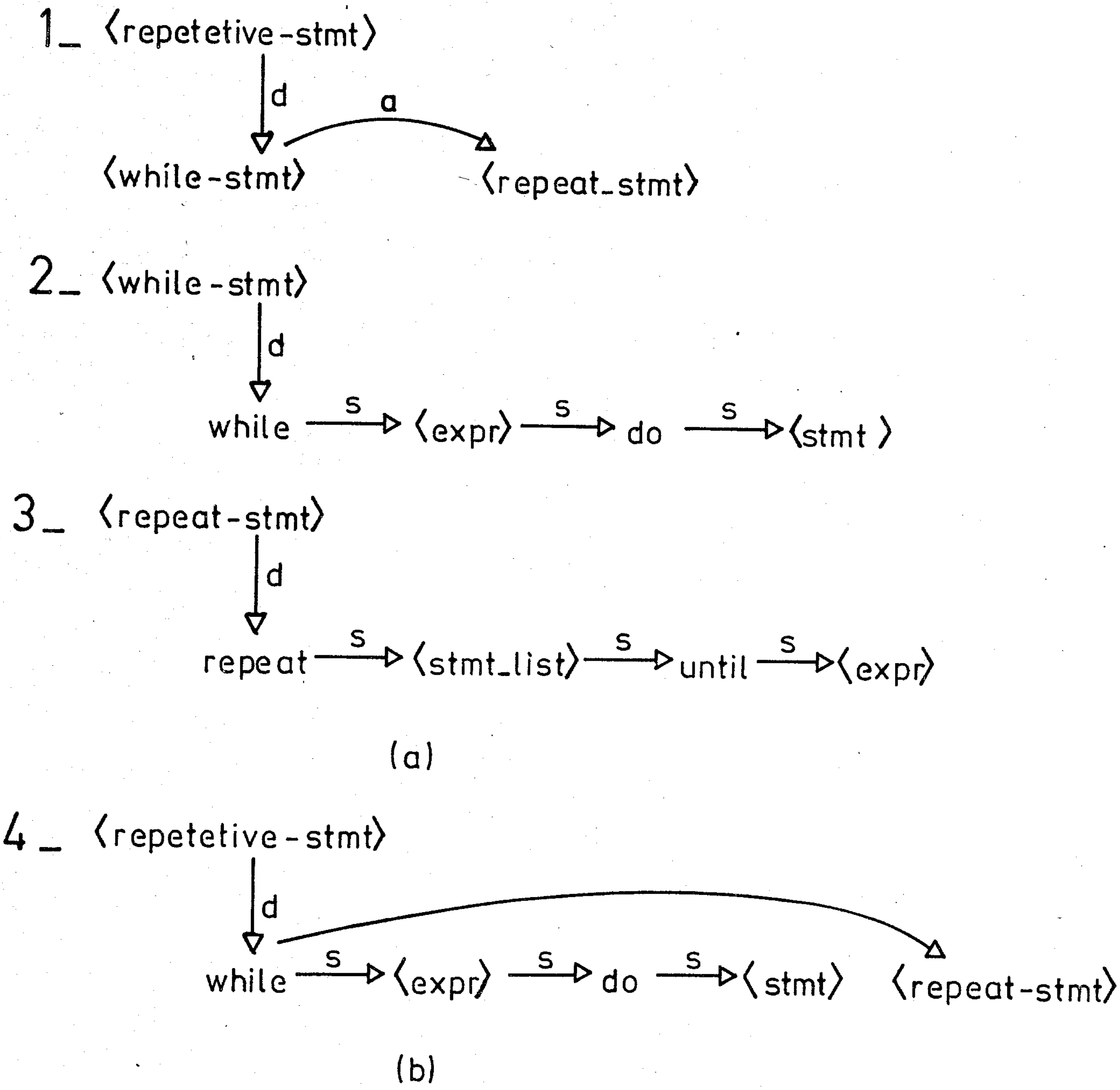
Backups can be avoided if we impose the restriction, NO NON-TERMINAL CAN HAVE AN ALTERNATE LINK DIFFERENT FROM FAIL VALUE, on the grammar. Thus in case of match fails, alternate link will be referred. If an alternate element exists we proceed with the alternate link. If its value is OK (which means expansion is completed) we move a higher level. But if its value is FAIL, an ERROR routine will be called.

Consider the graph in figure 3.5a. The definition of <repetetive-stmt> does not obey the above stated restriction, since the non-terminal <while-stmt> has an alternate link. Restriction is obeyed, by substituding the definition of <while-stmt> into the definition of <repetetive-stmt> (figure 3.5b).

These sort of substitution transformations on the grammar may lead to very long rules. But it is worth to do so, since only by this way backup problem can be solved

d) Representing empty strings

Consider a rule like <x> ::= a|b|c|d|e|ε where ε denotes the empty string. The presence of an empty string can be represented graphically by setting the alternate link of last element to OK value (figure 3.6).



(a)

(b)

Figure 3.5- An example of Substitution Transformation on a graph.

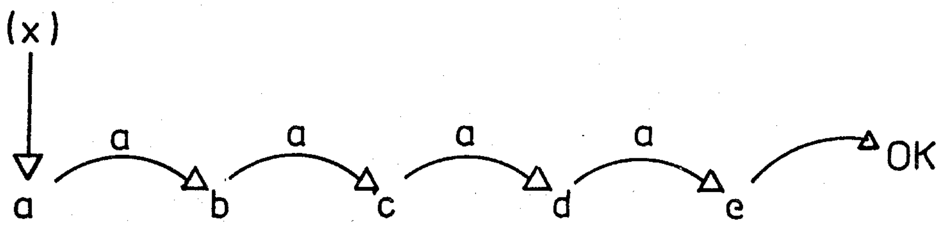


Figure 3.6- Graph of a Rule Containing an Empty String.

Recall that OK value in an alternate link, was used to indicate end of expansion while discussing iterative notation. OK value is used similarly here. Thus, if the alternate link of the terminal has been tried, that means expansion is completed with the empty string.

Representations of empty strings in this way may be time consuming, since all alternate links up to the last terminal should be tried. There is no simple way to get rid of empty strings and sometimes they are unavoidable. Yet some tricks can be used to get rid of them. For example consider the following rules.

```
<A> ::=  [<B>]
<B> ::=  c | <D>
<D> ::=  e | ε
```

These rules can be modified as follows;

```
<A> ::=  [<B>
<B> ::=  ] | c ] |  <D>]
<D> ::=  e
```

That would be easier and faster to parse. Graphical notations of both grammars are given in figure 3.7a and figure 3.7b in that order. Let input be, w = [] . In figure 3.7a when parsing this input, to see <B> is empty we have to move down to the definition of <D>. Then we check alternate link of the terminal e which is OK. Thus we move up to the first level to match].

In case of figure 3.7b, for the same input (w=[]) it is the second level and first element where a match is found.

3. Data Structures to Represent the Grammar In Core

PARSE uses two static tables to represent the grammar in core.

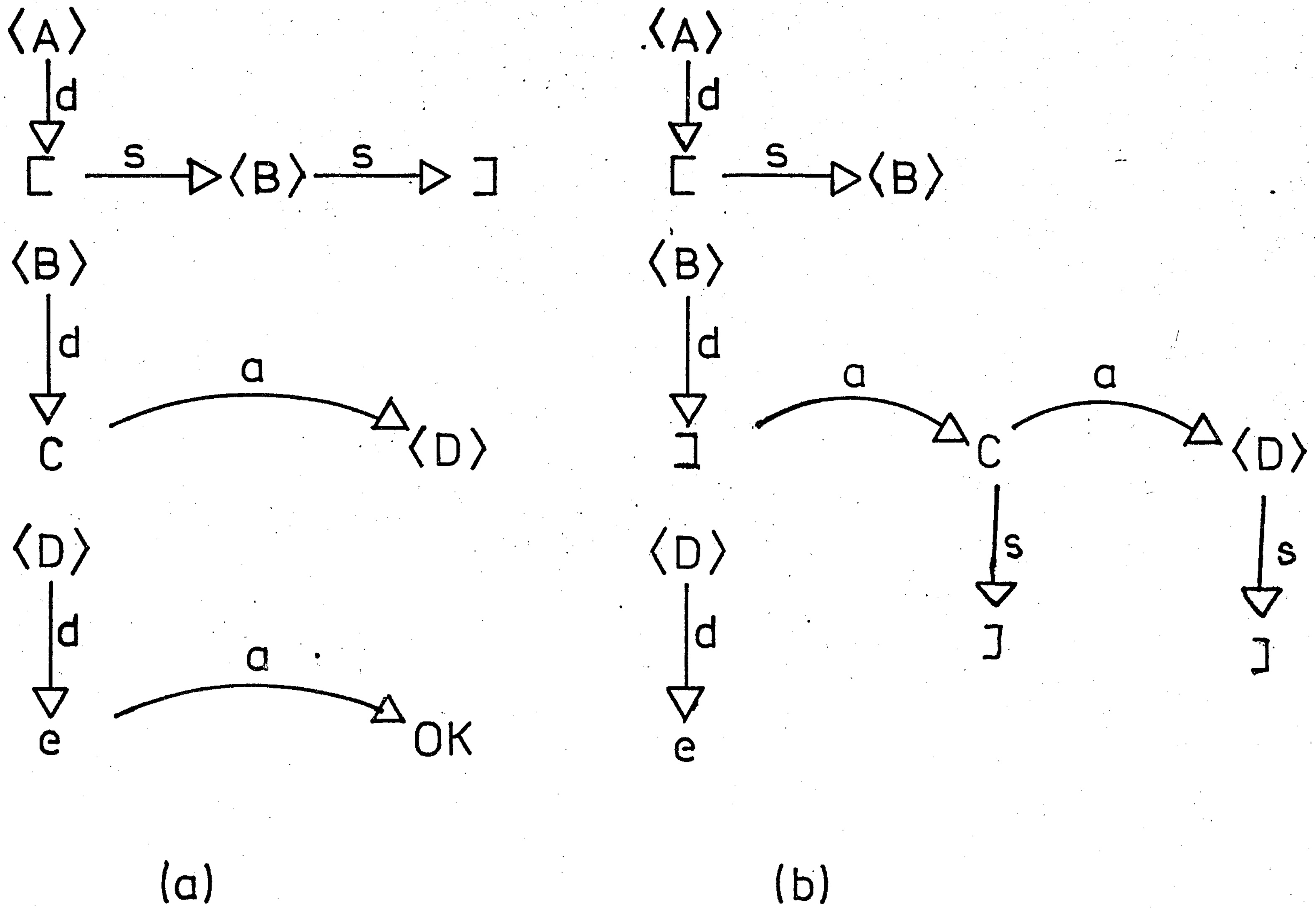


Figure 3.7. An Example to Get Rid of Empty Strings

i) SYNTAX TYPE TABLE (TTABLE)

TTABLE has an entry for each element of the vocabulary It is declared as;

```
type TPRANGE = 1..TPSIZE ; {≠elements in vocabulary}
var  TTABLE : array [TPRANGE] of
                 record
                   case TERMINAL : BOOLEAN of
                     TRUE  : (CLASS   : SYMBOL )
                     FALSE : (POINTER : SPRANGE);
                   end;
```

Tagfield TERMINAL is set to TRUE, if the corresponding element is a terminal and it is set to FALSE if the corresponding element is a non-terminal. Field POINTER is used for non-terminals, and points to an entry in the structure table (second table) where we record the definition of each non-terminal. If the graph notation is considered, this variant corresponds to the arc labelled d.

On the other hand, field CLASS is defined for terminals and is used to communicate with SCAN. In other words, this entry records the internal code of the symbol required.

ii) SYNTAX STRUCTURE TABLE (STABLE)

This table is used represent the rules of the grammar. Thus it has an entry for each symbol of a production, and for all productions. The table is declared as;

```
type  SPRANGE = -1.. SPSIZE; {size of grammar}
const FAIL    = -1; OK = 0;
var   STABLE : array [SPRANGE] of
                 record
                   SUCCESSOR, ALTERNATE : SPRANGE;
                   TPTYPE    :   TPRANGE
                 end;
```

Fields SUCCESSOR and ALTERNATE are used as internal pointers in STABLE and they correspond to arcs labelled s and a in graph notation. As discussed before, ALTERNATE field can take FAIL and OK values, and SUCCESSOR field can take OK value.

So subrange type SPRANGE has a lower bound equal to -1. Field TPTYPE is a backward pointer to TTABLE. It is used to get information about the current element being tested.

PARSE also uses a stack to keep completed parts of the parse tree being constructed. Two procedures (POP and PUSH) are used to manipulate this stack. An entry is allocated each time a non-terminal is met and deallocated each time the expansion of a non-terminal is completed.

4. Parsing Algorithm

Table 3.1 gives the algorithm of PARSE. Variable SPINX is used to index STABLE. It is initially set to 1, that is, it initially points to the first rule of the grammar. SPINX values are pushed on to the stack as long as they index non-terminal entries.

When a terminal is reached in the definition, it is compared with the input symbol. If they match procedure SUCCESS is called. Otherwise ALTERNATE field is tested. If its value is FAIL then an ERROR routine is called. If its value is OK then, this means end of expansion and again procedure SUCCESS is called. If none of these are true then SPINX is set to the value of ALTERNATE field.

Procedure SUCCESS is called in two cases. Its formal parameter MODE is set to 0, if a match occurs and is set to 1 if value of ALTERNATE link is OK. Function of SUCCESS is to advance SPINX to the successor of current element. It also tests whether expansion is completed by testing MODE = 1 or successor of the current element is OK. If so, it moves up a

TABLE 3.1- Parsing Algorithm

```
PROCEDURE PARSE ;

   PROCEDURE  POP                                 BEGIN  ....  END ;
   PROCEDURE  PUSH (ITEM : SPRANGE)     ;         BEGIN  ....  END ;
   FUNCTION NONTERMINAL : BOOLEAN       ;         BEGIN  ....  END ;

   PROCEDURE SUCCESS (MODE : INTEGER ) ;
   BEGIN

      WHILE (STABLE [SPINX].SUCCESSOR=OK) OR (MODE=1) DO
         BEGIN
            MODE := 0 ;

            IF STACKEMPTY  THEN
               IF TOKEN.CLASS = EOFPGM THEN
                  BEGIN
                     WRITELN ('..ACCEPTED..') ;
                     HALT
                  END
               ELSE
                  BEGIN
                     WRITELN ('**END OF ANALYSIS..',
                       '..NO MORE SYMBOLS ACCEPTED. ') ;
                     HALT
                  END ;

            SPINX := STACK [STACKTOP] ;

            POP   (* MOVE UP A LEVEL *)

         END ;     (* WHILE *)

      SPINX := STABLE [SPINX].SUCCESSOR

   END ;     (* SUCCESS *)


BEGIN          (* PARSE    *)

   SPINX := 1 ;   (*  INITIALIZE ROOT NODE *)

   WHILE TRUE DO

      WITH STABLE [SPINX] DO BEGIN

         WHILE NONTERMINAL(TPTYPE) DO BEGIN
            PUSH (SPINX)    ;
            SPINX := TTABLE[TPTYPE].POINTER
         END ;

         IF TTABLE[TPTYPE].CLASS = TOKEN.CLASS
         THEN    (* CURRENT ELEMENT = INPUT SYMBOL *)
            BEGIN
               SCAN ;    (*  GET NEXT TOKEN *)
               SUCCESS (0)
            END
         ELSE IF ALTERNATE = FAIL THEN
            BEGIN   ERROR  ; RECOVARY END
         ELSE IF ALTERNATE = OK THEN
              SUCCESS (1)
         ELSE SPINX := ALTERNATE

   END    (*   FOREVER LOOP *)

END.
```

level by a POP operation on stack.

Function NONTERMINAL returns true if its argument points to a non-terminal TTABLE entry. STACK-EMPTY is assumed to be a global variable set true by the procedure pop when the stack becomes empty.

Table 3.2a gives an example grammar, where as table 3.2b and 3.2c gives the internal representation of this grammar in core. A trace of the algorithm, for this grammar using input W = A > B * C ≠ is given in table 3.3. The input symbol ≠, is used to indicate end of string. In this table, existence of two functions SUCC and ALT are assumed which return successor and alternate values of their parameters.

E. Conclusion

Procedure PARSE is a predictive parse technique. Predictive parsers are efficient ways of implementing recursive descent parsing by handling the stack of activation records explicitly, and by keeping a set of tables to represent the grammar in core.

PARSE uses two static tables (TTABLE and STABLE) for grammar representation. Both tables have the prefix PACKED which requests a compact storage representation for values of the tables. Sizes of these tables are 97 words and 544 words respectively. Thus the space they occupy is not much.

The complete syntax of PASCAL written in BNF obeying the restrictions stated in this section is given in appendix C.

Various parsing techniques are explained in detail in GRIES (2) and AHO-ULLMAN (1).

TABLE 3.2- Parse Tables for an Example Grammar

GRAMMAR:

```
1) <expression>   ::= <simple exp> RELOP <simple exp>
2) <simple exp>   ::= <term> ADDOP <term>
3) <term>         ::= <factor> MULOP <factor>
4) <factor>       ::= IDENT | (<expression>)
```

(a)

INDEX	EXPLAINATION	TERMINAL	POINTER	CLASS
1	<expression>	FALSE	1	
2	<simple exp>	FALSE	4	
3	<term>	FALSE	7	
4	<factor>	FALSE	10	
5	RELOP	TRUE		RELOP
6	ADDOP	TRUE		ADDOP
7	MULOP	TRUE		MULOP
8	IDENT	TRUE		IDENT
9	(	TRUE		LPARENT
10	)	TRUE		RPARENT
11	≠	TRUE		EOFPGM

Contents of TTABLE

(b)

TABLE 3.2- Continued.

INDEX	EXPLANATION	SUCCESSOR	ALTERNATE	TPTYPE
EXPRESSION				
1	<simple exp>	2	FAIL	2
2	RELOP	3	OK	5
3	<simple exp>	2	FAIL	2
SIMPLE EXPRESSION				
4	<term>	5	FAIL	3
5	ADDOP	6	OK	6
6	<term>	5	FAIL	3
TERM				
7	<factor>	8	FAIL	4
8	MULOP	9	OK	7
9	<factor>	8	FAIL	4
FACTOR				
10	IDENT	OK	11	8
11	(	12	FAIL	9
12	<expression>	13	FAIL	1
13	)	OK	FAIL	10

Contents of STABLE

(c)

TABLE 3.3- Trace of Parsing Algorithm Using tables of TABLE 3.2

STEP	SPINX	IS TERMINAL?	OPERATION	STACK
1.	1	No	1. PUSH (SPINX) 2. SPINX := 4	→ 1
2.	4	No	1. PUSH (SPINX) 2. SPINX := 7	→ 4 1
3.	7	No	1. PUSH (SPINX) 2. SPINX := 10	→ 7 4 1
4.	10	Yes	1. $\overset{\text{IDENT}}{\text{Current element}} \overset{?}{=} \overset{\text{IDENT}}{\text{input symbol}}$ 2. Yes, 2.1. SCAN {input symbol := RELOP} 2.2. SUCCESS (0) 3. In SUCCESS with MODE = 0 3.1. (Succ (10) $\overset{?}{=}$ OK)OR(MODE$\overset{?}{=}$1) 3.2. Yes, 3.2.1. STACK not empty. POP 3.2.2. SPINX := 7 3.3. SPINX := SUCC(7) = 8 3.4. RETURN	→ 4 1
5	8	Yes	1. $\overset{\text{MULOP}}{\text{Current element}} \overset{?}{=} \overset{\text{RELOP}}{\text{input symbol}}$ 2. No, ALT(8) $\overset{?}{=}$ FAIL 3. No, ALT(8) $\overset{?}{=}$ OK 4. Yes, SUCCESS (1) 5. In SUCCESS with MODE = 1 5.1. (MODE $\overset{?}{=}$ 1) OR (SUCC(8) $\overset{?}{=}$ OK) 5.2. Yes, 5.2.1. STACK not empty, POP. 5.2.2. SPINX := 4 5.3. SPINX := SUCC(4) = 5 5.4. RETURN	→ 1

TABLE 3.3- Continued

STEP≠	SPINX	IS TERMINAL?	OPERATION	STACK
6.	5	Yes	1. $\overset{\text{ADDOP}}{\text{Current element}} \overset{?}{=} \overset{\text{RELOP}}{\text{input symbol}}$ 2. No, ALT(5) $\overset{?}{=}$ FAIL 3. No, ALT(5) $\overset{?}{=}$ OK 4. YES, SUCCESS(1) 5. In SUCCESS with MODE = 1 5.1. (MODE $\overset{?}{=}$ 1) OR (SUCC(5)$\overset{?}{=}$OK) 5.2. Yes, 5.2.1. STACK not empty, POP 5.2.2. SPINX := 1 5.3. SPINX := SUCC(1) = 2 5.4. RETURN	→
7.	2	Yes	1. $\overset{\text{RELOP}}{\text{Current element}} \overset{?}{=} \overset{\text{RELOP}}{\text{input symbol}}$ 2. Yes, 2.1. SCAN {input symbol := ident} 2.2. SUCCESS (0) 3. In Success with MODE = 0 3.1. (Succ (2) $\overset{?}{=}$ OK) or (MODE $\overset{?}{=}$ 1) 3.2. No, SPINX := SUCC (2) = 3 3.3. RETURN	
8.	3	No	1. PUSH (SPINX) 2. SPINX := 4	→ 3
9.	4	No	1. PUSH (SPINX) 2. SPINX := 7	→ 4 3
10.	7	No	1. PUSH (SPINX) 2. SPINX := 10	→ 7 4 3

TABLE 3.3- Continued

STEP≠	SPINX	IS TERMINAL?	OPERATION	STACK
11.	10	Yes	1. $\overset{\text{IDENT}}{\text{Current element}} \overset{?}{=} \overset{\text{IDENT}}{\text{input symbol}}$	
			2. Yes	
			2.1. SCAN input symbol := MULOP	
			2.2. SUCCESS(0)	
			3. In SUCCESS with MODE = 0	
			3.1. (SUCC(10) $\overset{?}{=}$ OK)OR(MODE$\overset{?}{=}$1)	
			3.2. Yes,	
			3.2.1. STACK not EMPTY, POP	→ 4
			3.2.2. SPINX := 7	3
			3.3. SPINX := SUCC(7) = 8	
			3.4. RETURN	
12.	8	Yes	1. $\overset{\text{MULOP}}{\text{Current element}} \overset{?}{=} \overset{\text{MULOP}}{\text{input symbol}}$	
			2. Yes,	
			2.1. SCAN {input symbol := IDENT}	
			2.2. SUCCESS(0)	
			3. In SUCCESS with MODE = 0	
			3.1. (SUCC(8)$\overset{?}{=}$OK)OR(MODE$\overset{?}{=}$1)	
			3.2. No, SPINX := SUCC(8) = 9	
			3.3. RETURN	
13.	9	No	1. PUSH (SPINJ)	→ 9
			2. SPINX := 10	4
				3
14.	10	Yes	1. $\overset{\text{IDENT}}{\text{Current elmt}} = \overset{\text{IDENT}}{\text{input symbol}}$	
			2. Yes,	
			2.1. SCAN {input symbol}:≠	
			2.2. SUCCESS(0)	
			3. In SUCCESS with MODE = 0	
			3.1. (SUCC(10)$\overset{?}{=}$OK)OR(MODE$\overset{?}{=}$1)	→ 4
			3.2. Yes,	3
			3.2.1. Stack not empty, POP	
			3.2.2. SPINX := 9	
			3.3. SPINX := SUCC(9) = 8	
			3.4. RETURN	

TABLE 3.3- Continued

STEP≠	SPINX	IS TERMINAL?	OPERATION	STACK
15	8	Yes	1. $\overset{\text{MULOP}}{\text{Current element}} \overset{?}{=} \overset{\neq}{\text{input symbol}}$ 2. No, ALT(8)$\overset{?}{=}$FAIL 3. No, ALT(8)$\overset{?}{=}$OK 4. Yes, SUCCESS(1) 5. In SUCCESS with MODE = 1 5.1. (SUCC(8)$\overset{?}{=}$OK)OR(MODE$\overset{?}{=}$1) 5.2. Yes, 5.2.1. Stack not empty, POP 5.2.2. SPINX := 4 5.3. SPINX := SUCC(4) = 5 5.4. RETURN	→ 3
16	5	Yes	1. $\overset{\text{ADDOP}}{\text{Current element}} \overset{?}{=} \overset{\neq}{\text{input symbol}}$ 2. No, ALT(5)$\overset{?}{=}$FAIL, 3. No, ALT(5)$\overset{?}{=}$OK, 4. Yes, SUCCESS(1) 5. In success with MODE = 1 5.1. (SUCC(5)$\overset{?}{=}$OK)OR(MODE$\overset{?}{=}$1) 5.2. Yes, 5.2.1. STACK not empty, POP 5.2.2. SPINX := 3 5.3. (SUCC(3)$\overset{?}{=}$OK)OR(MODE$\overset{?}{=}$1) 5.4. Yes, 5.4.1. STACK is empty, 5.4.2. Input symbol$\overset{?}{=}$≠ 5.4.3. Yes, 5.4.3.1. PRINT('ACCEPTED'); 5.4.3.2. HALT 5.5. RETURN	→

IV. ERROR RECOVERY

A. Definition

Programs submitted to a translator often have errors of various kinds. A good translator, therefore, should find as many errors as possible.

While discussing parsing in the previous chapter, our purpose was to determine whether a sentence of a given language was accepted or not. It was assumed that detection of the first error stops parsing with an "UNACCEPTED SENTENCE" message. Yet, as stated here, even in the presence of errors, a translator should be able to continue parsing and scan the entire program trying to analyze all of it. The term "error recovery" is used for the process of determining how to continue analyzing a source program when an error is found.

Errors are classified as syntactic and semantic errors. Syntactic errors are those detectable by the lexical or syntactic phase of a translator. Other errors detectable by a translator are classified as semantic errors. In this thesis semantic errors are limited to errors of declaration. Lexical phase errors are outlined in section 3 . Their recovery is simple and performed by SCAN. Recovery of semantic errors, here, is defined as to suppress extra error messages and is described in Section 5. So this section, describes the algorithm used to recover syntactic errors.

Recovery algorithm modifies the input, so the correct portions of the program can be pieced togather and succesfully processed.

B. Syntactic Errors

Typical of these are the following;

- the insertion of an extranous symbol
- the deletion of a required symbol
- the replacement of a correct symbol by an incorrect symbol.
- the transposition of two adjacent symbols

Note that a replacement error and a transposition error can each be treated as special cases of an insertion error followed by a deletion error.

Here are a few common examples of syntactic errors.

1. Missing right parenthesis

```
MIN(A,2*(3+B);
```

2. Missing semicolon

```
A:=3
B:=4;
```

3. ":" in place of ";" or "=" in place of ":="

```
A:=3:                A = 3;
B:=4;                B:=4;
```

4. Misspelled keyword

```
PORCEDURE A;
```

5. Extra blank

```
(*  COMMENT  * )
```

(1) and (2) are examples of deletion errors, (3) replacement error (5) an insertion error and (4) a transposition error.

Quite frequently we can not detect that an error has occured until long after it has taken place. For example, consider the PASCAL program fragment

```
FORI := 1+K[20] TO 100 DO L:=L+1;
```

The obvious error is a missing blank between the keyword FOR and the name I. No error is discernible, however, until the keyword TO has been read, since PARSE treats FORI:=1+K[20] as an assignment statement.

This example shows that the detection of an error may occur an arbitrarily long distance after the place where the error actually occurred.

Error recovery strategy tries to change the small portion of the program containing the error into a string that is legal, by making minimum number of insertions, deletions and symbol modifications necessary. Because of the distance problem, recovery algorithm may generate several error messages for a single error. If we do happen to generate a few, it doesn't really matter.

C. Recovary From Syntactic Errors

At any point of a parse of a source program, the program has the form

xTt

where x represents the part already processed, T is the next symbol to be scanned, and t is the rest of the program. Suppose an error occurs with T. In the TOP-DOWN method, this means that the partial tree built to cover x can not be extended to cover T.

At this point we must determine how to change the program to "fix" the error. It can be changed most easily in the following ways (or perhaps combinations of them.).

1. Delete T and try to parse again
2. Insert a string of terminals q between x and T (yielding xqTt) and begin parsing using the head of

qTt. This insertion should allow us to process all of qT before another error occurs.

3. Delete some symbols from the tail of x.

Deleting part of x, we must change the semantic information accordingly, and this is not easy to do so. Methods (1) and (2) will be our main methods of recovering.

In figure 4.1 the incomplete branch named P corresponds to an application of the rule <P>::=<A>; and ";" is the incomplete part of the branch. Similarly, the incomplete branch named E corresponds to an application of the rule <E>:=<T>$\{+<T>\}_0$ To complete the branch, we need a single <T>. followed by a number of "+<T>"s. The incomplete part is therefore <T>{+<T>}.

Incomplete parts of branches play a large role in error recovery; they tell us, in effect, what can or should appear next in the source program.

Now let us suppose that an error occurs during a parse; no partially constructed syntax tree can further be built. The following recovery algorithm is performed.

1. A list L of the symbols in the incomplete parts of the incomplete branches is constructed.
2. The head symbol of Tt is repeatedly examined and discarded (yielding a new string Tt) until a T is found such that $U \Rightarrow * \ldots T \ldots$ for some U in L (either U=T or $U \Rightarrow + \ldots T \ldots$).
3. An incomplete branch which caused U of step 2 to be put in L is determined.
4. A terminal string q is determined such that if inserted just before T, the continuation of the parse will cause T to be correctly linked to the incomplete branch of step 3.
5. q is inserted just before T and the parse is continued,

beginning with the head symbol of q as the incoming symbol.

Consider, for example, the parse as indicated by fig. 4.1. An error has occurred with ")" as the incoming symbol. We have L={<T>,+,;}. By step 2 we see That <T>==>+...)... That is, <T>==><F>==>(<E>). This notation can be expressed as ")" is reachable from the nonterminal <T>.

D. Implementation

1. Formal Definition of the Algorithm

I will illustrate the algorithm using the following grammar

<P>::=<A> ;
<A>::= i:=<E>
$<E>::=<T>\{+<T>\}_o$
$<T>::=<F>\{*<F>\}_o$
<F>::= i|(<E>)

At ony step of a parse, one or more syntax trees have been constructed, with some incomplete branches. An incomplete branch named U corresponds,to an application of a rule

$$U::=x_1x_2 \ldots x_{i-1}x_i \ldots x_n$$

where $x_1 \ldots x_{i-1}$ is the completed part of the branch and $x_i \ldots x_n$ the incomplete part of the branch. Figure 4.1 gives the tree for the sentence i:=i+); of the grammar, stated above. In this figure solid lines show a a partially completed tree while the dotted lines show how the branches named <P> and <E> might be completed.

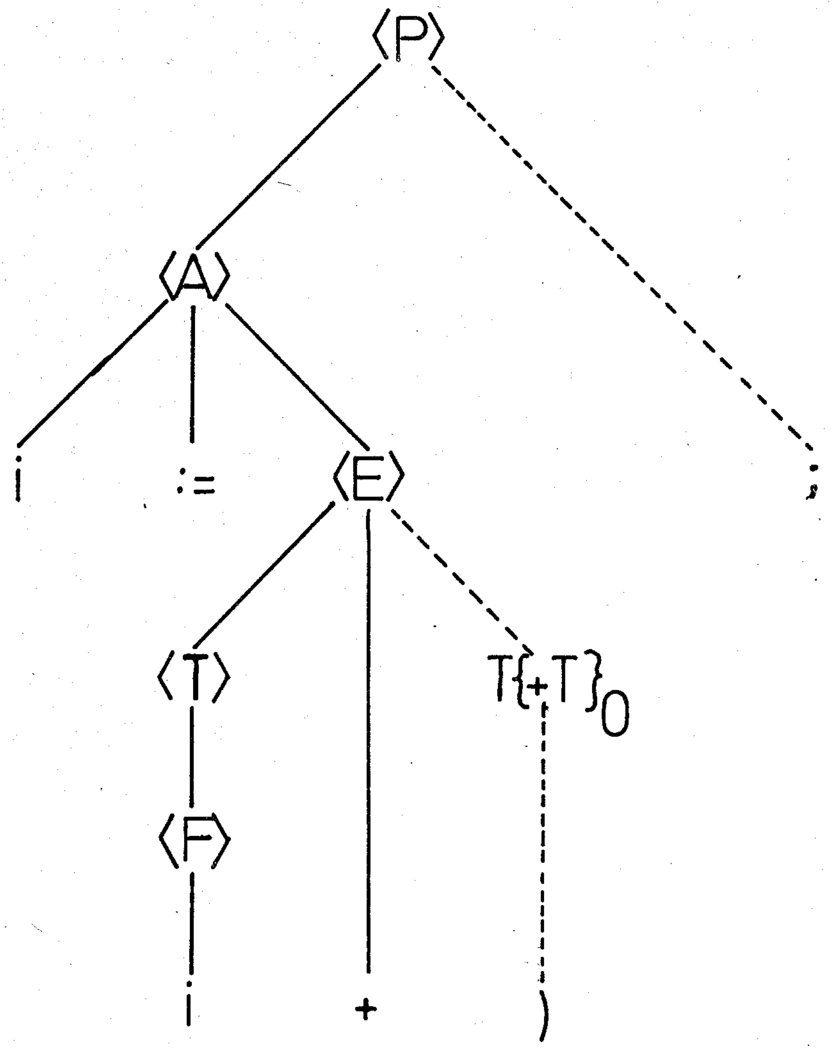


Figure 4.1- Top Down Tree Before Error Recovery.

The incomplete branch which caused <T> to be put in L is <E>::=T{+T}$_0$ We must therefore insert a string q to complete this branch. To cause the ")" to be associated with this branch, q must include (<E>. Since q will be a string of terminals <E> must be expanded. Simplest expansion of <E> is the symbol i. Therefore q will be (i. Thus, in step 5 qT becomes (i) beginning with head symbol "(" and this recovers the error (Figure 4.2).

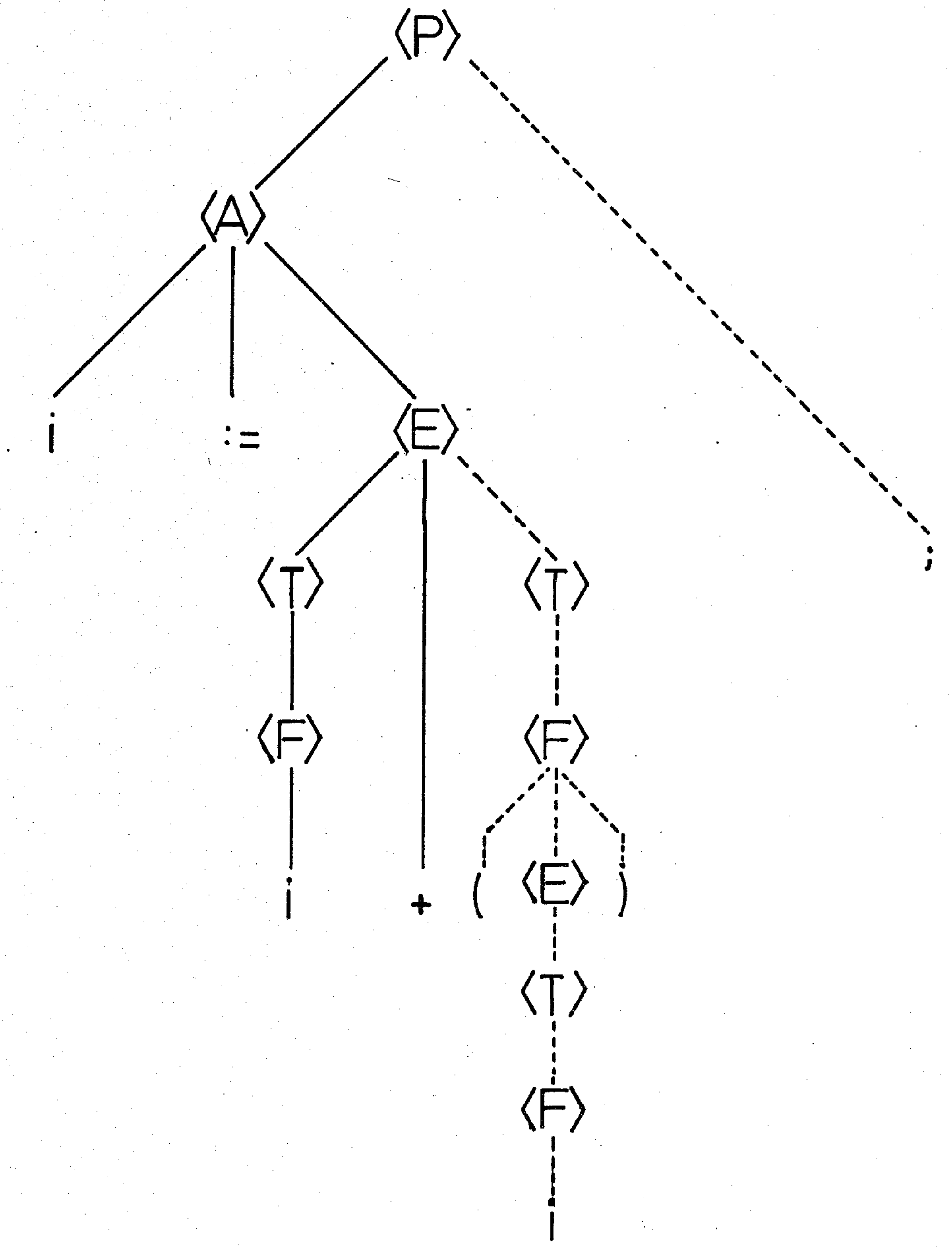


Figure 4.2. Top Down Free After Error Recovery

Figure 4.3. illustrates how the algorithm works with the input i:=(i+);. In this example L={;,),+,<T>}. This time ")" is immediately selected by step 2 (U=")").

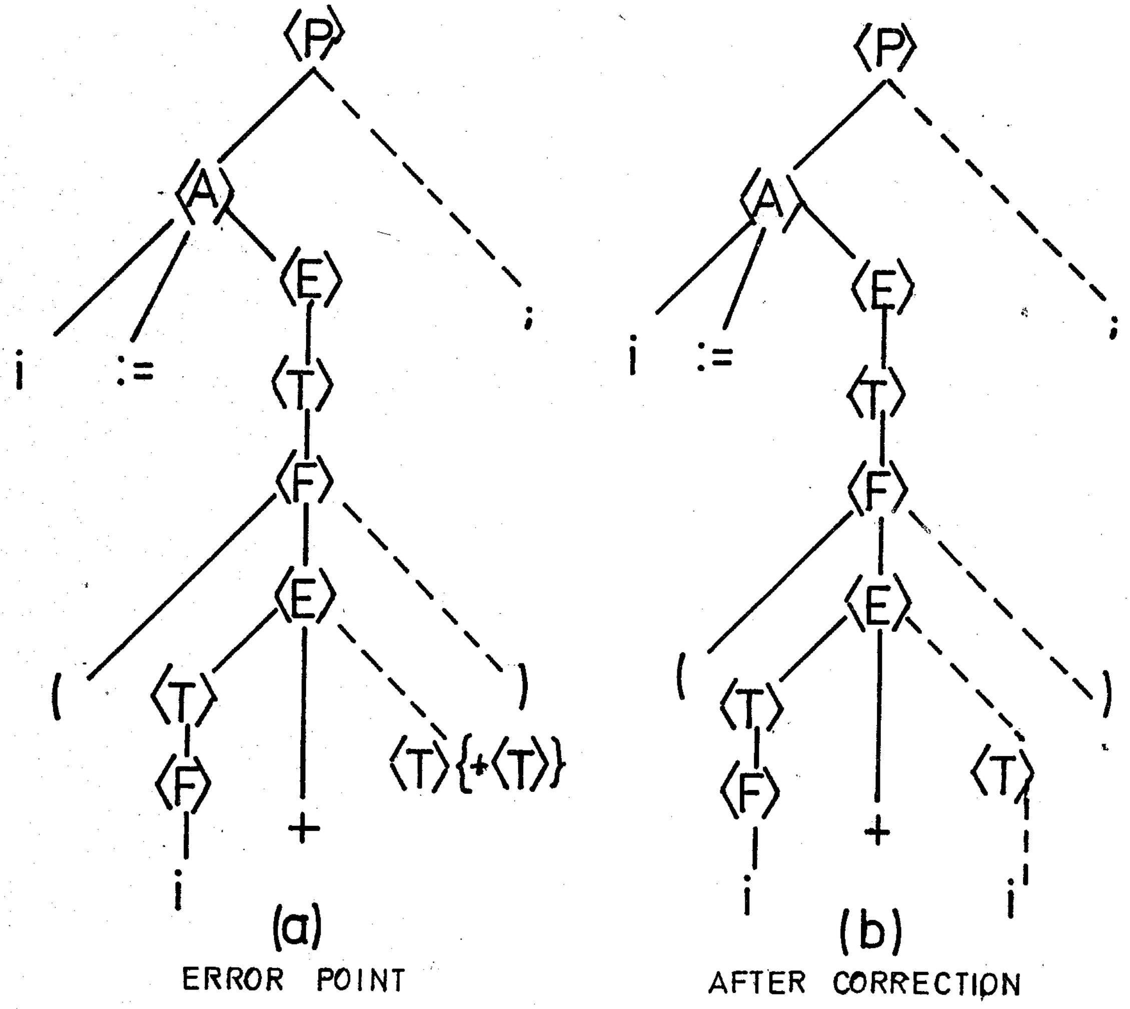


Figure 4.3- Top Down Error Recovery

The incomplete branch which caused ")" to be put in L is <F> ::=(<E>). To cause ")" to be associated with this branch, we must insert a string to complete the branch <E> ::=<T>{+<T>}$_o$. Recovery algorithm always tries to insert simplest string possible, and this again is identifier i.

2. Recovery Algorithm

The recovery technique described in this section is quite applicable to our parsing algorithm, since we already have incomplete branches kept in our parse stack.

Step 1 of the recovery algorithm requires preparing a list of the symbols in the incomplete parts of the incompleted branches, to be used by step 2. By the help of the parse stack it is easy to prepare this list, but do we need such a list explicitly? Actually no! Using iteration, incompleted parts can be reached and transmitted to step 2 one by one. Yet, since it's convenient to assume the existence of such a list, we do it so.

Purpose of step 2, is to search the list, for some member U, such that $U \Longrightarrow * ...T...$ where T corresponds to the last scanned source symbol. It is possible that no member U, satisfying this condition exists. Then we just discard T, get next source symbol and repeat the process with this symbol as being T. Finally, the desired member U will be reached. A BOOLEAN function named FIND, is declared to perform step 2. FIND is repeatedly called for each member of the list, until a member satisfying the above condition is found.

Search of step 2 starts from current STABLE position (or SPINX) to test symbols of "local" context. Then stack must be searched (incomplete branches) to determine symbols in the "global" context, starting from top of the stack, going downwards since it is most probable that the error is due to an incomplete branch near the stacktop.

Steps 1 and 2 are combined into a single step and expressed algorithmically in table 4.1. In this algorithm FOUND is a bolean variable and STIND is an integer variable used to index STACK. As in table 3.3, SUCC is assumed to be a function that returns successor of its parameter.

Every source symbol discarded means an illegal symbol so an associated ERROR routine is called. When we exit from search loop, STIND indicates the incomplete branch which accepts last scanned token, T as a member of its definition. Thus step 3 of the formal algorithm is also covered.

TABLE 4.1- Recovery Algorithm--Part 1

```
REPEAT
   FOUND := FIND(SUCC (SPINX)) ; (* TEST LOCAL CONTEXT *)
   STIND := STACKTOP ;
         (******TEST GLOBAL CONTEXT*****)
   WHILE (NOT FOUND) AND (STIND >0) (* STACK NOT EMPTY *) DO
      BEGIN
         FOUND := FIND (SUCC (STACK[STIND])) ;
         STIND := STIND -1
      END  ;

   IF NOT FOUND THEN  (* DISCARD T AND GET NEXT TOKEN *)
      BEGIN
         ERROR(6) ;
         SCAN      (* GET NEXT TOKEN *)
      END
UNTIL FOUND ;
```

So, we come to step 4 of the initial algorithm where insertion string q is determined. On exit from the loop of table 4.1. first, the value of STIND is tested. If its value is less than the value of STACKTOP, that means there are incomplete branches in between which must be manually completed. In other words, there are non-terminals indicated by STACK entries between STIND and STACKTOP, each of which, requires a terminal string generated for them. This test is implemented as a loop which calls a procedure, named STRING, repeatedly (Table 4.2).

TABLE 4.2- Recovery Algortihm--Part 2

```
STEMP   := STACKTOP ;

WHILE STIND < STEMP DO
   BEGIN
      STRING (STABLE [STACK[STEMP]].TPTYPE) ;
      STEMP := STEMP - 1
   END ;
```

STRING is a procedure that receives a non-terminal as its parameter. It generates simplest possible terminal string for the non-terminal denoted by its parameter and catanates it to q.

Treatment of the incomplete branch where the error is found (the STACK entry indexed by STIND) will be different. In this case a string will be generated (and catenated to q) until the last scanned token T, is reached in the definition, unlike the way STRING works where, the string is generated until the expansion is completed. This last part is accompolished by a call to a routine named INSERT. This routine receives successor of the STABLE entry indexed by STACK[STIND].

When q is completely determined, every symbol in it indicates a missing symbol in the position where the error is detected. So for every symbol in q, an error message of the form "......EXPECTED" is printed.

3. Description of Routines

i) FUNCTION FIND

This function receives as input a structure table (STABLE) entry. It will try to reach the last scanned source symbol (or T) by moving through the links associated qith its input. While doing this, it must visit all the successors and alternates of its input. But if its input is a non-terminal it must look to its definitions successors and alternates also and so on. So function FIND is designed as a recursive routine.

Calling sequence of FIND is based on its input and defined as follows.

```
if input is a NON-TERMINAL
    call itself with DEFINITION of input;
    call itself with SUCCESSOR of input

else
    call itself with ALTERNATE of input;
    call itself with SUCCESSOR of input
```

By this way all the successors and alternates for an STABLE entry can be visited. Table 4.3. gives the complete algorithm of FIND. Here parameter SPINX is the input value. Since there are three distinc calls of FIND there must be three stopping criterias to control them;

a) When calling FIND with SUCC(SPINX) as argument, if SUCC(SPINX)=OK or SPINX>SUCC(SPINX), that indicates end of successors for SPINX, so call must be prohibited.

b) When calling FIND with ALT(SPINX) as argument, if ALT(SPINX)=OK or ALT(SPINX)=FAIL, that means there are no alternates for SPINX, so call is not performed.

TABLE 4.3- Algorithm FIND

```
FUNCTION FIND (SPINX : INTEGER) : BOOLEAN ;
    VAR FOUND : BOOLEAN ;
BEGIN
    FOUND  := FALSE  ;
    WITH STABLE [ SPINX ], TTABLE [ TPTYPE ] DO
        CASE TERMINAL OF
            FALSE  :
                BEGIN
                    IF  UNMARKED (FINDCONTRL,TPTYPE) THEN
                        FOUND        :=*FIND (POINTER) ;
                    IF  NOT FOUND THEN
                        IF (SUC  <> OK) AND (SUC   > SPINX) THEN
                            FOUND   := FIND (SUC )      ;
                END  ;

            TRUE   :
                BEGIN
                    IF CLASS = TOKEN.CLASS
                    THEN FOUND   := TRUE
                    ELSE
                        BEGIN
                            IF  (ALT <> FAIL) AND (ALT <> OK) THEN
                                FOUND    := FIND (ALT)   ;
                            IF  NOT FOUND THEN
                                IF (SUC <> OK) AND (SUC> SPINX) THEN
                                FOUND    := FIND (SUC )
                        END
                END
        END  (* CASE *)   ; FIND := FOUND
 END  ;     (* FIND *)
```

c) Calling FIND with the definition of a non-terminal may cause infinite loops, since the grammar is recursively defined (directly or indirectly). Consider a rule like <exp>::=i|(<exp>). Call of FIND with <exp> may cause another call of FIND with <exp> and so on, since <exp> contains <exp> in its definition.

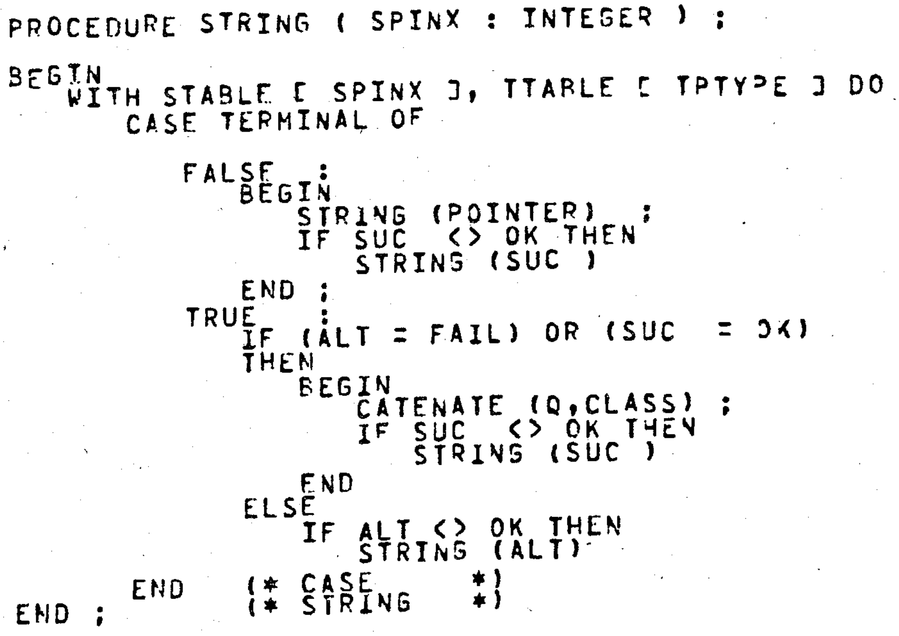
Thus, in order not to refer a non-terminal more than once, a boolean function UNMARKED is defined which returns true if its argument has already been tried.

ii) PROCEDURE STRING

This procedure receives an STABLE entry corresponding to a non-terminal as its input. It will try to generate simplest possible terminal string for its input. Procedure STRING is also recursively defined, since definition of its input may also contain non-terminals.

When a successor link with value OK is met (which indicates end of expansion) or an alternate link with OK value is tried (which shows optionality), the execution of STRING will be terminated. Table 4.4. shows the algorithm of STRING. To catenate the current symbol to the end of q, a procedure named CATENATE is defined.

TABLE 4.4- Algorithm STRING

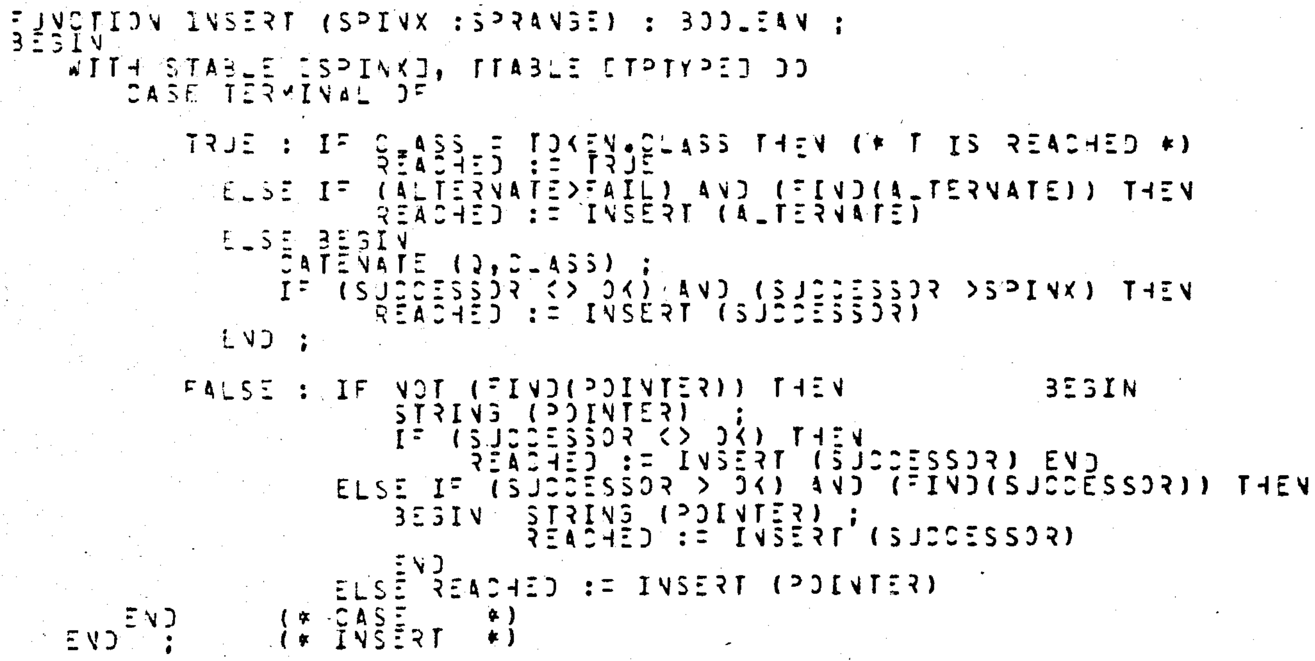


```
PROCEDURE STRING ( SPINX : INTEGER ) ;
BEGIN
   WITH STABLE [ SPINX ], TTABLE [ TPTYPE ] DO
      CASE TERMINAL OF
         FALSE :
            BEGIN
               STRING (POINTER) ;
               IF SUC <> OK THEN
                  STRING (SUC )
            END ;
         TRUE :
            IF (ALT = FAIL) OR (SUC  = OK)
            THEN
               BEGIN
                  CATENATE (Q,CLASS) ;
                  IF SUC  <> OK THEN
                     STRING (SUC )
               END
            ELSE
               IF ALT <> OK THEN
                  STRING (ALT)
      END    (* CASE     *)
END ;        (* STRING   *)
```

iii) FUNCTION INSERT

This routine is the last portion of the recovery algorithm. It receives the STABLE entry, that contains last scanned source symbol, T in its definition as input. INSERT is recursively defined as the previous two routines. It generates a terminal string and catenates it to q. Execution of INSERT will terminate when T is reached. INSERT is designed as a boolean function. If it returns false that means a compiler error exists. Table 4.5. gives the algorithm of INSERT. This algorithm is a simplified one. Actual algorithm is much more complex, to take care of infinite loops that frequently occur due to recursive definition of the grammar. For actual algorithm refer to appendix A.

TABLE 4.5- Algorithm INSERT



```
FUNCTION INSERT (SPINX :SPRANGE) : BOOLEAN ;
BEGIN
   WITH STABLE [SPINX], TTABLE [TPTYPE] DO
      CASE TERMINAL OF

         TRUE : IF CLASS = TOKEN.CLASS THEN (* T IS REACHED *)
                    REACHED := TRUE
           ELSE IF (ALTERNATE>FAIL) AND (FIND(ALTERNATE)) THEN
                    REACHED := INSERT (ALTERNATE)
           ELSE BEGIN
              CATENATE (Q,CLASS) ;
              IF (SUCCESSOR <> OK) AND (SUCCESSOR >SPINX) THEN
                    REACHED := INSERT (SUCCESSOR)
           END ;

         FALSE : IF NOT (FIND(POINTER)) THEN             BEGIN
                    STRING (POINTER)  ;
                    IF (SUCCESSOR <> OK) THEN
                       REACHED := INSERT (SUCCESSOR) END
                 ELSE IF (SUCCESSOR > OK) AND (FIND(SUCCESSOR)) THEN
                    BEGIN  STRING (POINTER) ;
                           REACHED := INSERT (SUCCESSOR)
                    END
                 ELSE REACHED := INSERT (POINTER)
      END        (* CASE    *)
   END  ;        (* INSERT  *)
```

Table 4.6 gives the trace of the RECOVERY algorithm using the grammar of table 3.2. Input is assumed to be w=A>B*)#. This sentence is analyzed exactly same as table 3.3. until step 13. So table 4.6. outlines steps after 13.

E. Conclusion

Error recovery requires, we can continue analyzing without too much possibility of generating several error messages for a single error. If we do happen to generate few, it does not realy matter. This can be provided by keeping the insertion string q, as small as possible. So procedure STRING and FUNCTION insert are designed, to minimize the length of q.

Sometimes, recovery algorithm causes extra errors in the incoming source symbols, because symbols of q may conflict with the actual purpose of the programmer.

The recovery algorithm used in this thesis is described in GRIES(2).

TABLE 4.6- Trace of Recovery Algorithm Using Parse Tables of Table 3.2.

STEP#	SPINX	IS TERMINAL	OPERATION	INPUT	STACK
13	9	NO	1. PUSH(SPINX)	A>B*)# ↑	9 4 3
			ident ')'		
14	10	Yes	1. current elmt=input symbol		
			2. No, ALT(10)$\overset{?}{=}$FAIL		
			3. No, ALT(10)$\overset{?}{=}$OK		
			4. No, SPINX:=ALT(10)=11		
			( )		
	11	Yes	1. current elmt$\overset{?}{=}$input symbol	A>B*)# ↑	
			2. No, ALT(11)$\overset{?}{=}$FAIL		
			3. Yes, ERROR, RECOVER		
			4. In RECOVER with SPINX=11		
			4.1. FIND(succ(11))		
			4.2. In FIND with SPINX=12		
			4.2.1. Is TERMINAL(12)		
			4.2.2. No, UNMARKED (TPTYPE(12))		
			4.2.3. Yes, FIND(1)		
			4.3. ın FIND with SPINX=1		
			⋮ ↓		
	10		4.4. In FIND with SPINX=10		
			4.4.1. Is TERMINAL(10)		
			4.4.2. Yes, current elmt$\overset{?}{=}$input symbol		
			4.4.3. No, ALT(10) <> OK and ALT (10) <> FAIL?		
			4.4.4. Yes, FIND (ALT(10))		
	11		4.5. In FIND with SPINX=11		
			4.5.1. Is TERMINAL(11)		
			( )		
			4.5.2. Yes, current elmt$\overset{?}{=}$input symbol		
			4.5.3. No, ALT(11) <> OK and ALT (11) <> FAIL ?		

TABLE 4.6- continued

STEP#	SPINX	IS TERMINAL	OPERATION	INPUT	STACK
			4.5.4. No, SUCC(11) <> OK and 11<(SUCC(11)=12)?		
			4.5.5. Yes, FIND(SUCC(11))		
	12		4.6. IN FIND with SPINX=12		
			4.6.1. Is TERMINAL(12)		
			4.6.2. No, UNMARKED (TPTYPE(12))		
			4.6.3. No, SUCC(12) <> OK OR 12 <SUCC(12)=13?		
			4.6.4. Yes, FIND(SUCC(12))		
	13		4.7. In FIND with SPINX=13		
			4.7.1. Is TERMINAL(13)		
			))		
			4.7.2. Yes, current elmt$\overset{?}{=}$input symbol		
			4.7.3. Yes, FOUND:=TRUE		
			4.7.4. FIND:=TRUE {FIND RETURNS TRUE}		
			4.8. STIND := STACKTOP		
			4.9. q := empty string		
			4.10. REACHED:=INSERT(11)		
	11		4.11. In INSERT with SPINX=11		
			4.11.1. Is TERMINAL(11)		
			( )		
			4.11.2. Yes, Current symbol$\overset{?}{=}$input symbol		
			4.11.3. No, ALT(11) FAIL		
			4.11.4. No.		
			4.11.4.1. q:=CATENATE(q,"(")		
			4.11.4.2. (SUCC(11) <> OK) and (12>11)?		
			4.11.4.3. Yes, REACHED:= INSERT(12)		

TABLE 4.6- continued

STEP#	SPINX	IS TERMINAL	OPERATION	INPUT	STACK
	12		4.12. In INSERT with SPINX=12		
			4.12.1. Is TERMINAL(12)		
			4.12.2. No, FIND(12)?{refer operation 4.1}		
			4.12.3. Yes, SUCC(12) <> OK		
			4.12.4. Yes, FIND(SUCC(12))		
	13		4.12.5. In FIND with SPINX=13 FIND returns true, immediate success		
	1		4.12.6. In STRING with SPINX=1		
	1		4.12.6.1. Is TERMINAL(1)		
			4.12.6.2. No, STRING(5)		
	5		4.12.7. In STRING with SPINX=5		
			4.12.7.1. Is TERMINAL(5)		
			4.12.7.2. No, STRING(7)		
	7		4.12.8. In STRING with SPINX=7		
			4.12.8.1. Is TERMINAL(7)		
			4.12.8.2. No, STRING(10)		
	10		4.12.9. In STRING with SPINX=10		
			4.12.9.1. Is, TERMINAL(10)		
			4.12.9.2. Yes, (ALT(10)=FAIL)or(SUCC(10)=OK)?		
			4.12.9.3. Yes,q:=q ||IDENT=(IDENT ⋮ ↓ return form STRING ↓		
	12		4.2.10. REACHED:=INSERT(SUCC(12))		
	13		4.13. In INSERT with SPINX=13))		
			4.13.1. Current elmt=input symbol		
			4.13.2. Yes, return from INSERT with input	A>B*(ident) ↑ head of q	
16	11		1. current element=input symbol		
			2. Yes ...		9 4 3

V. SYMBOL TABLE

A. Definition

A translator needs to collect and use information about names appearing in the source program. This information is entered into a data structure called a SYMBOL table. The information collected about a name includes the string of characters by which it is denoted, its type, its structure etc.

Each time a name is encountered, the symbol table is searched to see whether that name has been seen previously. If it is new, it is entered into the symbol table. Information about a name is entered, by syntactic analyzer while parsing declarations.

Information collected in the symbol table, is used in semantic analysis, (that is, in checking uses of identifiers are, consistent with their declarations) and in code generation.

Symbol table can be used to aid in error detection and correction. For example, we can record whether an error message such as "variable A undefined" has been printed out before, and refrain from doing so more than once.

In block structured languages the same identifier can be used to represent distinct names with nested scopes. In such languages, the symbol table mechanism must make sure that the innermost occurence of an identifier is always found first and that names are removed from the active portion of the symbol table when they are no longer active.

Symbol table mechanism, thus should allow us;

1. Determine whether a given name is in the table,
2. add a new name to the table,
3. access the information associated with a given name,
4. add new information for a given name,
5. delete a name or groups of names from the table.

B. Symbol Table Organizations

This section describes the ways of representing symbol tables in general.

1. Unsorted and Sorted Tables

The easiest way to organize a table is to add entries in the order they arrive. A search requires N/2 comparisons on the average, for a match if N is greater than 20, and this is inefficient.

Searching can be performed more efficiently if the table entries are sorted according to string of characters denoting the name. Efficient search techniques such as binary search can be used in this case.

Another method of accessing symbols in a table is using hash-addressing. This is a technique for converting symbols to indexes of entries in the table (the indexes are numbered 0,1,2,...,N-1 where the table has N entries). The index is obtained by "hashing" the symbol, i.e. by performing some simple arithmetic or logical operation on the symbol. As long as, two symbols do not hash to the same index, we have no problem. Trouble occurs, however if two symbols hash to the same index. This is called a collision, and the hash algorithm must take care of it.

2. Block Structured Tables

Algol-like languages have a nested block and procedure structure. The same identifier may be declared and used many times in different blocks and procedures and each such declaration must have a unique symbol table entry associated with it. Given an identifier, the problem is then to discover the correct symbol table entry for it.

The rule is to look first in the current block, then the surrounding block and so on, until a declaration of that identifier is found. Such a search can be implemented by keeping all the symbol table entries for each block contigious, and by using a block list (Figure 5.1).

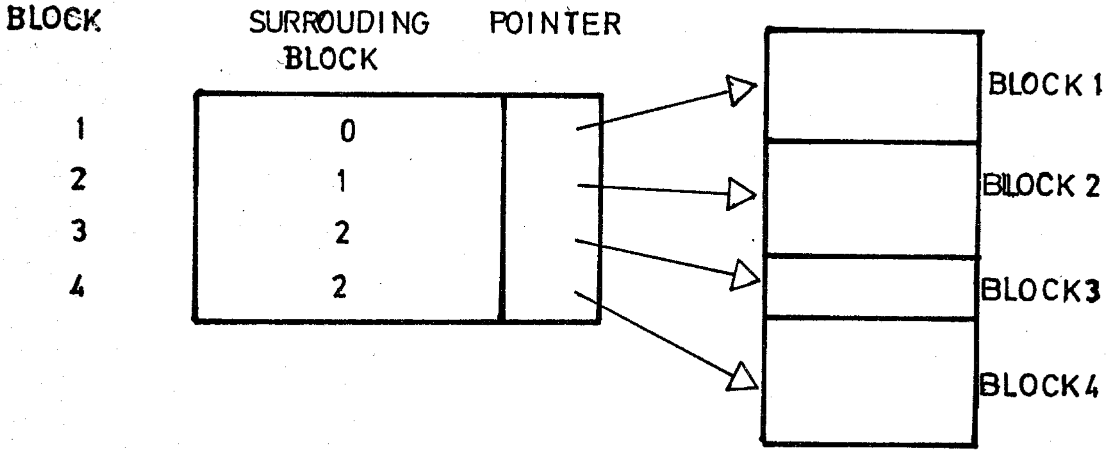


Figure 5.1- Block Structure.

Once an associated block is found, searching it would be simple.

3. Tree Structured Tables

This organization strategy is used in the implementation. The method uses a binary tree to order the entries. Each node of the tree represents a filled entry of the table, the root node being entry 1. Figure 5.2a shows the table with one entry for identifier G. Suppose now that the identifier D is to be entered. A branch is drawn for it to the left, since D < G (figure 5.2b).. Now let the M be entered. Since G < M a branch is drawn for it to the right from G (figure 5.2c). Finally let the identifier E be entered. E < G, so we travel down the left branch from G, and to the right of D (figure 5.2b). Figure 5.2e shows the tree after identifiers A, B and F have been added in that order.

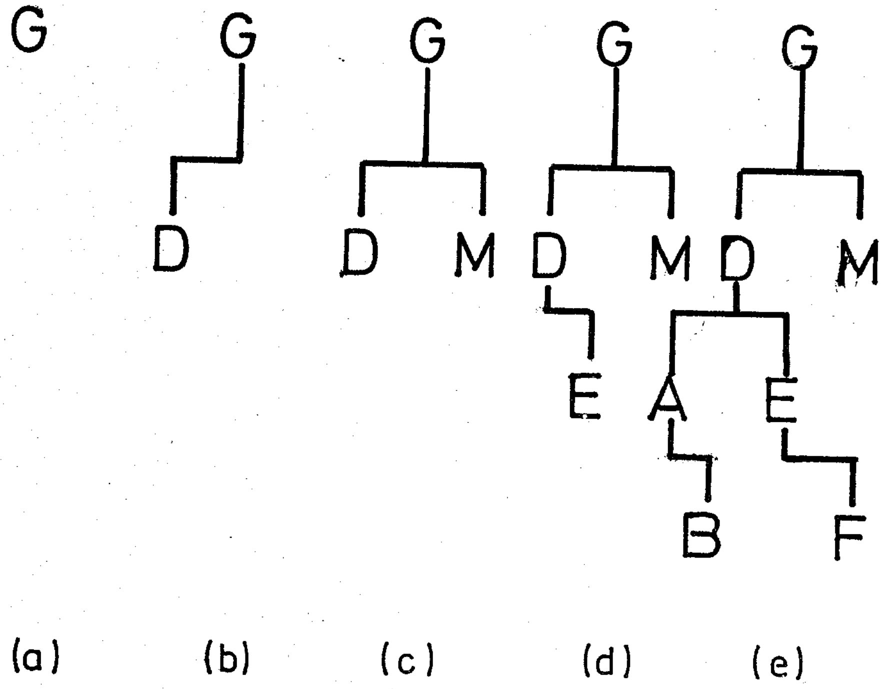


Figure 5.2. Binary Tree Illustration.

One can implement this, by having two pointer fields with each entry, one for the left and one for the right branches.

C. Implementation

In this section data structures and routines used for symbol table manipulation in this thesis will be explained.

1. Organization of Symbol Table

As mentioned before symbol table is organized in a tree structured manner. Since, PASCAL has a nested procedure structure, each procedure has its own tree structure to store its local variables. In other words, symbol table is organized as an unbalanced biniary tree at each level.

It is also necessary to access variable of surrounding procedure (i.e. global variables). This means access to the trees of higher levels. So, keeping trace of levels (with their tree pointers) is necessary. This is accompolished by a stack (Section 6.3.4, describes implementation of this stack).

2. Data Structures to Represent the Symbol Table
   There are two Basic Tables

   i) STRUCTURE: Used to keep information about the attributes of variables. That is, it describes the structure of data types.

   ii) IDNTFR : Holds the identifiers declared throughout the program.

Dynamic allocation facilities and variant records of PASCAL are ideal to represent these tables.

Consider table 5.2. Possible structures of PASCAL data types are given by the enumareted type STRUCTFORM. Similarly IDCLASS shows classes of identifiers.

Field NAME of record IDNTFR is used to keep the actual symbol itself and it is used as a search key. Fields LLINK, RLINK in the same record are left and right branch pointers of the tree. However left and right links are not sufficient. Sometimes sequential linking becomes necessary, for example in a parameter list (or in a user defined scalar) where the parameters (scalar identifiers) must be linked in the order which they are declared. IDTYPE is the STP pointer indicating the structure of the identifier recognized.

Variants of IDNTFR which are dependent on the tagfield KLASS are introduced to indicate level counters, to differrentiate actual-formal declarations and to state values of constants.

TABLE 5.2- Data Structures for Symbol Table

```
CONST MAXLEVEL = 20   ;

TYPE

    STRUCTFORM  = (SCALAR,SUBRANGE,POINTER,POWER,ARRAYS,
                   RECORDS,PARAMLIST,FILES,TAGFIELD,VARIANT);
    DECLKIND    = (STANDARD,DECLARED)                           ;

    STP         =  ^STRUCTURE                                   ;
    IDP         =  ^IDNTFR                                      ;

    STRUCTURE   =  PACKED RECORD
                      CASE FORM : STRUCTFORM OF
                         SCALAR : ( CASE SCALKIND : DECLKIND OF
                                    DECLARED : (FSTCONST : IDP);
                                    STANDARD : () ) ;
                         SUBRANGE :( RANGETYPE     : STP ;
                                     MIN,MAX       : INTEGER )  ;
                         POINTER  :( ELTYPE        : STP )      ;
                         POWER    :( ELSET         : STP )      ;
                         ARRAYS   :( PACKD         : BOOLEAN    ;
                                     INXTYPE,AELTYPE :STP )     ;
                         RECORDS  :( RPACK         : BOOLEAN    ;
                                     FSTFLD        : IDP        ;
                                     RECVAR        : STP )      ;
                         PARAMLIST:( FSTPAR        : IDP )      ;
                         FILES    :( FILTYPE       : STP )      ;
                         TAGFIELD :( TAGFIELDP     : IDP        ;
                                     TAGTYPE       : STP        ;
                                     FSTVAR        : STP )      ;
                         VARIANT  :( NXTVAR,SUBREC : STP        ;
                                     VARVAL        : INTEGER )  ;
                   END  ;

    IDCLASS =
       (TYPES,KONST,VARS,FIELD,PARAMS,FUNC,PROC,PROG) ;
    IDKIND    = (ACTUAL,FORMAL) ;
    LEVRANGE = 0 .. MAXLEVEL  ;

    IDNTFR = PACKED RECORD
                NAME : PACKED ARRAY [1..12] OF CHAR ;
                IDTYPE          : STP ;
                NEXT            : IDP ;
                LLINK,RLINK     : IDP ;
                CASE  KLASS     : IDCLASS OF
                   KONST :     (VALUES : CONSTANT);
                   VARS  :     (VKIND  : IDKIND   ;
                                VLEV   : LEVRANGE);
                   PROC,FUNC :
                      (CASE PFDECLKIND : DECLKIND OF
                            STANDARD : () ;
                            DECLARED :
                               (PFLEV : LEVRANGE ;
                                PARAMPTP : STP ;
                                CASE PFKIND : IDKIND OF
                                   ACTUAL : (FORWDECL : BOOLEAN)))
             END  ;
```

Now consider STRUCTURE record. Here a tagfield of type STRUCTFORM is used. If the FORM is (user defined) SCALAR we need to know IDNTFR pointer of the first enumareted constant, (field FSTCONST). If it is SUBRANGE, then we need to know RANGETYPE (it must be SCALAR) which points to an another STRUCTURE entry. Fields MIN and MAX contain the lower and upper bounds of the range respectively. When the FORM is ARRAYS we need to know if it is packed, how it is indexed and type of its element. Multi-dimensional arrays are represented as arrays of arrays, so there is no need to have a field indicating how many dimensions the array has.

Other variants except for RECORDS, I believe are clearly understandable from table 5.2. Record structures must have their own trees like procedures, since fields of a record are in accessible if the name of the record is not specified. Thus field FSTFLD of the variant RECORDS should point to the root node of the associated tree. Field RECVAR is used to point to the TAG information if the record contains variants otherwise it is NIL.

A record type may be considered as a road map to an area of memory. It defines how the memory is to be interpreted. A variant record type provides several different road maps for the same area of memory, and a tag field value determines which road map is currently in use. So it is reasonable to consider each variant as a subrecord, activated according to the value of tag field. (If there is no tag field any of these subrecords can be activated arbitrarily).

Field TAGFIELDP of variant TAGFIELD is an IDNTFR pointer indicating the tag, and is NIL if there is no tag field. Field TAGTYPE indicates the type of tag which must be a scalar type. Field FSTVAR points to the first variant of the record.

TABLE 5.3- Symbol Table Dump for a Variant Record

```
     TYPE   A = RFCORD
                  N,ZZ : RFAL ;
                  CASE M : INTEGER OF
                     1,2 : ( K : REAL ) ;
                       3 : ( J : INTFGER )
               END ;

**** IDP1 @ :

            NAME = 'A'      ; IDTYPE = STP1 ;    KLASS = TYPES ;

**** STP1 @ :

            FORM = RECORDS ; FSTFLD = IDP2    ; RECVAR= STP2   ;

**** IDP2 @ :

            NAME = 'N'      ; IDTYPE = REALPTR ; KLASS = FIELD ;

**** IDP3 @ :

            NAME = 'ZZ'     ; IDTYPE = REALPTR ; KLASS = FIELD ;

**** STP2 @ :

            FORM     = TAGFIELD; TAGFIELDP = IDP4 ;
            TAGTYPE = INTPTR   ; FSTVAR    = STP3 ;

**** IDP4 @ :

            NAME = 'M'      ; IDTYPE = STP2    ; KLASS = FIELD ;

**** STP3 @ :

            FORM = VARIANT ; NXTVAR = STP4 ;
            SUBREC = STP5  ; VARVAL = 1    ;

**** STP4 @ :

            FORM = VARIANT ; NXTVAR = STP6 ;
            SUBREC = STP5  ; VARVAL = 2    ;

**** STP5 @ :

            FORM = RECORDS ; FSTFLD = IDP5 ; RECVAR = NIL  ;

**** IDP5 @ :

            NAME = 'K'      ; IDTYPE = REALPTR ; KLASS = FIELD ;

**** STP6 @ :

            FORM = VARIANT ; NXTVAR = NIL  ;
            SUBREC = STP7  ; VARVAL = 3    ;

**** STP7 @ :

            FORM = RECORDS ; FSTFLD = IDP6 ;
            RECVAR = NIL   ;

**** IDP6 @ :

            NAME = 'J'      ; IDTYPE =  INTPTR ; KLASS = FIELD ;
```

Field SUBREC of variant VARIANT points to the subrecord which will be activated if the tag field value is equal to the value of the field VARVAL. (VARVAL is unused if there is no tag field). Eventually field NXTVAR points to the next variant of the record if there are any.

To understand fully the representation of a variant record it is necessary to examine Table 5.3. In this example, variables IDP1 to IDP6 are assumed to be pointer constants of type IDP, whereas variables STP1 to STP7 are pointer constants of type STP. Similarly INTPTR and REALPTR are STP pointer constants, pointing to the definition of standard PASCAL types INTEGER and REAL (Figure 5.2 gives the linked list representation of table 5.3).

3. Predefined PASCAL Identifiers

The following identifiers are entered to the tree of level 0, with their STRUCTURE definitions during initialization.

1. MAXINT
   is a constant whose value is dependent on the machine,
2. INTEGER
   is a standard scalar type. Its value ranges between -MAXINT ... MAXINT.
3. REAL
   is a standard scalar type. Its values are an implementation dependent finite subset of real numbers.
4. CHAR
   is a standard scalar type whose values are a set of implementation dependent characters.
5. BOOLEAN
   is a pre declared scalar type which is defined as
   <u>type</u> BOOLEAN = (FALSE, TRUE);

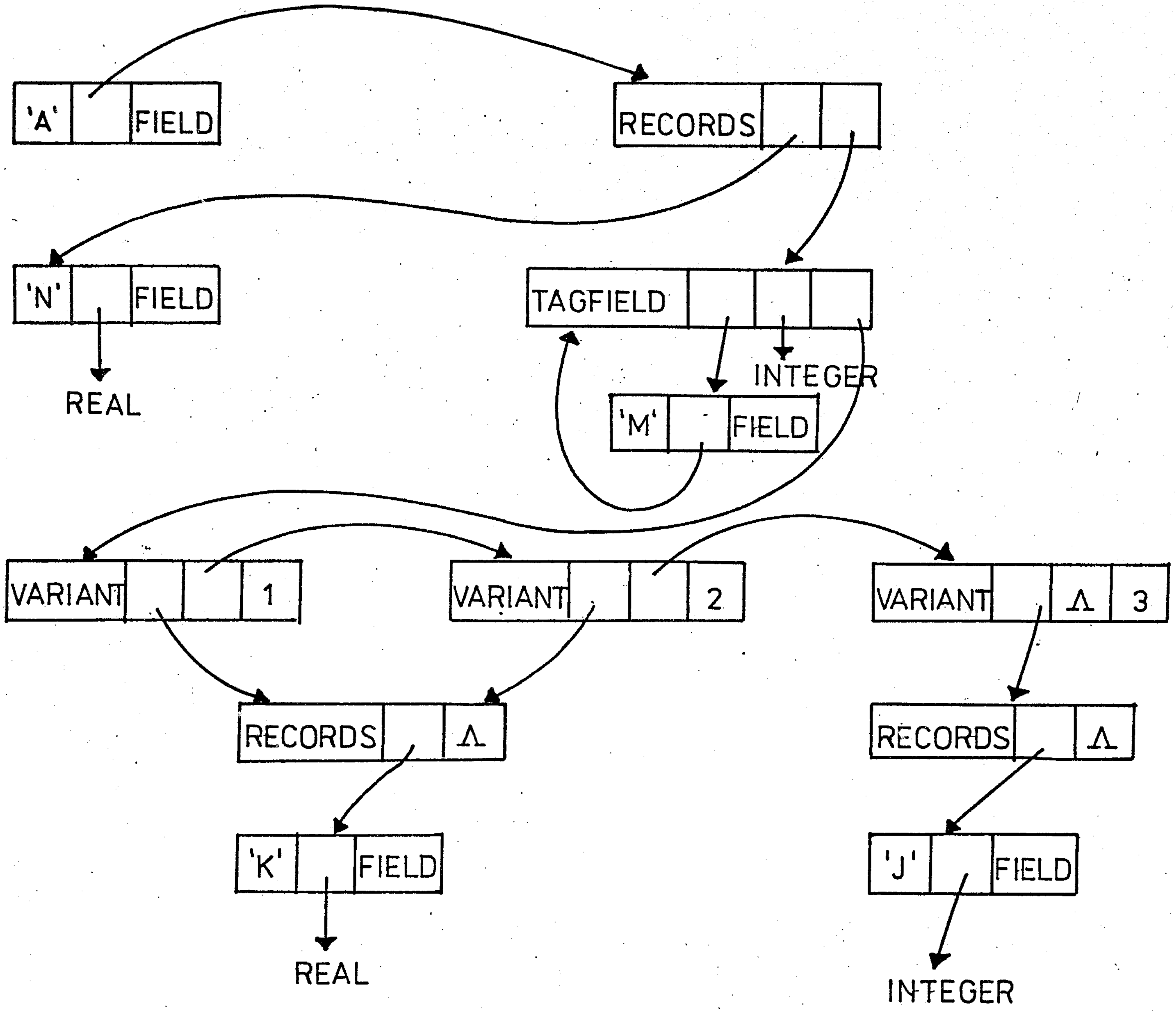


Figure 5.3- Linked List Representation of TABLE 5.3.

Therefore FALSE and TRUE are constants of type BOOLEAN enumarated as 0,1 in that order.

6. TEXT

   is a predefined file type which is declared as

   ```
   type TEXT = file of CHAR;
   ```

7. NIL

   is a implementation dependent constant used for pointers to indicate null entry.

8. OTHER IDENTIFIERS

   These include standard PASCAL procedures and functions such as READ, WRITE, EDLN, EOF etc.

4. Local Tree Pointers

A single dimensional array indexed by level counter is used as a stack, to point local trees. This stack is declared as;

```
var DISPLAY: array [0..20] of IDP:
    TOP    : 0..20; {level counter which is initially
                     zero}
```

TOP is incremented each time a procedure is met. First variable will be entered into the DISPLAY [TOP] . Other variables of the procedure will be entered (or searched) taking DISPLAY [TOP] as the root node.

Similarly TOP is decremented each time a procedure is terminated. Thus the tree of that level will be deallocated, since it won't be used anymore.

Current tree is pointed by DISPLAY [TOP] every time. Trees of surrounding procedures can be reached by referring DISPLAY entries whose indexes are lower than the value of TOP.

5. Symbol Table Manipulation Routines

1. PROCEDURE ENTERID (IDPTR:IDP);
   Enters identifier pointed by IDPTR (according to the algorithm stated in section 5.B.3) into the symbol table into the innermost level.

2. PROCEDURE SEARCHID (KLASS: set of IDCLASS; IDPTR:IDP);
   Searchs identifier pointed by IDPTR whose class is included in the set KLASS. The search includes whole symbol table, that is trees of all higher levels.

   If it fails that means an undeclared identifier is met. So SEARCHID enters it to the symbol table by calling a procedure DECLARE to suppress extra error messages.

3. PROCEDURE DECLARE (IDPTR:IDP);
   Used to suppress multiple printout of error messages as mentioned above.

4. PROCEDURE SEARCHSECTION (FCP:IDP; var FCP1=IDP);
   Searchs the tree whose root node is pointed by FCP and returns result in FCP1. This kind of search is necessary for instance in searching fields of a record or searching forward declared pointer types.

5. PROCEDURE ENTERSTDIDS;
   Used to enter predefined PASCAL identifiers at level 0.

6. PROCEDURE PRINTABLES;
   Dumps symbol table, separately at each level.

7. PROCEDURE SEMANTICS (ACTION:INTEGER);
   As mentioned before BUPASCAL makes semantic analysis at the declaration level. In other words it enters the identifiers declared throughout the program into the symbol table, keeps information about them and detects associated errors. Thus, purpose of this procedure is to perform these tasks. Calling sequences for the procedures ENTERID, SEARCHID and SEARCHSECTION is

controlled by this procedure.

SEMANTICS is called by the parser each time a terminal is scanned or a non-terminal is stacked (unstacked). Assume that the terminal IDENT is scanned and we are processing var declarations. It will be searched within the symbol table (PROCEDURE SEARCHID) to see if it is previously declared if so an error message is given otherwise it will be entered into the symbol table by calling procedure ENTERID. Successive calls of SEMANTICS will be, thus by terminals indicating the structure of the identifier being processed, for instance successive terminal might be the keyword ARRAYSY. So associated STRUCTURE record, can be modified to have FORM=ARRAYS.Similarly all attributes can be determined. SEMANTICS is called when a non-terminal is stacked, to set same flags, to take some initiative actions etc. or when a non-terminal is unstacked to reset some flags, to decrement a level counter etc.

SEMANTICS will be called using current STABLE entry (SPINX) as argument. Thus SPINX uniquely determines the action to be taken. For non-terminals, SPINX will be negated to indicate the non-terminal is unstacked. SEMANTICS uses a case statement based on its formal parameter ACTION, to perform necessary action. If a case label whose value is equal to the value of ACTION does not exist SEMANTICS will do nothing.

D. Conclusion

The symbol table mechanism of this implementation uses dynamic trees. Thus size of the symbol table is not restricted. On the other hand searching technique used (binary search) has an order of log N (where N is number of nodes in the tree) is quite efficient unless the number of nodes is very few.

GRIES (2) and AHO-ULLMAN (1) describes several symbol organization techniques. HOROWITZ-SAHNI (3) explains dynamic allocation and deallocation of binary trees.

Symbol table will be dumped if the option $PRINTABLES is specified. This dump is similar to that of table 5.3, except actual pointer constants are used. Variables local to a block (program or procedure) are printed just before the first "begin" is scanned (predefined identifiers are also included while printing variables of PROGRAM .. i.e. global variables).

VI. CONCLUSION

The finalized version of BUPASCAL has been successfuly tested on a number of input programs, four of which are given in the appendix. These sample inputs contain arbitrary compile time errors to test error recovery capability of BUPASCAL.

Even though BUPASCAL accepts programs written in STANDARD PASCAL as input, it is designed to accept any language whose grammar is given by parse tables. BUPASCAL uses two static tables (TTABLE and STABLE) for grammar representation. The size of TTABLE depends on the size of the vocabulary of the language under consideration, where as the size of STABLE depends on the number of rules used to define the grammar.

Therefore recovery algorithm is also designed, to respond to any language. Because of recursive and iterative representations of grammars, I believe recovery algorithm was the most important and difficult part of this thesis. Test runs showed that, even the most terrible errors can be recovered by the algorithm presented here.

BUPASCAL does not include semantic analysis at the statement level and code generation. But the design of parse tables allows subroutines (or coroutines) to be associated with the productions of the grammar. These routines are to be intended to perform semantic analysis and to generate intermediate code when called at appropriate times by the syntactic analyzer. When a severe error is detected, code generation terminates where as semantic analysis should continue until the whole program is processed. An associated routine is called each time a syntactic primitive (or terminal) is met and each time a stack operation is performed.

Procedure SEMANTICS defined in section 5 is designed in this way. In this procedure ACTION values correspond to the coroutines described above. This routine can be modified to include semantics analysis at the statement level and code generation by enlarging the range of ACTION values. (i.e. by increasing case labels to respond all entries of STABLE).

APPENDIX A
SOURCE PROGRAM

```
        $PAGE


(****************************************************************
*                                                               *
*                                                               *
*                                                               *
*)  PROGRAM BUPASCAL    (INPUT,OUTPUT) ;                      (*
*                                                               *
*              PURPOSE ..                                       *
*                 IMPLEMENTATION OF AN ANALYZER FOR PASCAL.     *
*                 STANDARD PASCAL IS USED.                      *
*                 INPUT, OUTPUT MODE IS FIELDATA.               *
*              DESIGNED BY ..                                   *
*                 CEM ATAC                                      *
*              COMPUTER ..                                      *
*                 UNIVAC 1106                                   *
*              DATE ..                                          *
*                 SEPTEMBER 16, 1982                            *
*              CENTER ..                                        *
*                 BOSPHOROUS UNIVERSITY COMPUTER CENTER         *
*                                                               *
*                                                               *
*                                                               *
****************************************************************)


    LABEL    1111  ; (* MAIN RETURN ADDRESS *)
    CONST

       CHARMAX      = 12 ;  BUFMAX      = 80 ;  MAXERR      = 10 ;
       FIRSTCHAR    = '@';  LASTCHAR    = ' ';  MAXSET      = 71 ;
       STRGLGTH     = 78 ;  SETSIZE     = 72 ;  NSETS       = 12 ;
       FAIL         = -3 ;  OK          = -2 ;  NOACT       =  0 ;
       SPSIZE       =268 ;  TPSIZE      = 93 ;
       MAXREAL      = 0.898846567431157951E+307   ;
       SQRTREAL     = 0.670390396497129853E+154   ;
       DISPLIMIT    = 20 ;   MINMIN     =-5000; MAXMIN      =5000;
       MAXLEVEL     = 20 ;

    TYPE

       SYMBOL =(          IDENT,      INTCONST,    REALCONST,   CHARCONST,
                    STRINGCONST,        EOFPGM,        MULOP,       ADDOP,
                          RELOP,       LPARENT,      RPARENT,    LBRACKET,
                       RBRACKET,         COMMA,    SEMICOLON,      PERIOD,
                         DOTDOT,         ARROW,        COLON,      ASGNOP,
                        ARRAYSY,       BEGINSY,       CASESY,     CONSTSY,
                           DOSY,      DOWNTOSY,       ELSESY,       ENDSY,
                         FILESY,         FORSY,    FORWARDSY,  FUNCTIONSY,
                         GOTOSY,          IFSY,      LABELSY,       NOTSY,
                           OFSY,      PACKEDSY,       PROCSY,   PROGRAMSY,
                       RECORDSY,      REPEATSY,        SETSY,      THENSY,
                           TOSY,        TYPESY,      UNTILSY,       VARSY,
                        WHILESY,        WITHSY,      OTHERSY     ) ;

       OPERATOR =
              (            ASTR,          RDIV,         IDIV,       ANDOP,
                           IMOD,          PLUS,        MINUS,        OROP,
                           LTOP,          LEOP,         GEOP,        GTOP,
                           NEOP,          EQOP,         INOP     ) ;

       STRGRANGE    = 0      .. STRGLGTH ;
       STRGINDEX    = 1      .. STRGLGTH ;
       SETRANGE     = 0      .. MAXSET   ;
       SPRANGE      =FAIL    .. SPSIZE   ;
       TPRANGE      =FAIL    .. TPSIZE   ;
       NNONTRMS     =52      .. 93       ;
       CHSIZE       = 0      .. 168      ;
       DISPRANGE    = 0      .. DISPLIMIT;

       CSTCLASS     = (INTGR, REEL, PSET, STRING, KHAR)              ;
       CHARSTRING   = PACKED ARRAY [STRGINDEX] OF CHAR               ;
       CHARARRAY    =        ARRAY [STRGINDEX] OF CHAR               ;

       CSTADDR      = @ CONSTANT  ;
       CONSTANT     = PACKED RECORD
                         INTVAL    : BOOLEAN                         ;
                         CASE CLASS : CSTCLASS OF
```

```
                   INTGR : (IVAL : INTEGER           )  ;
                   REEL  : (RVAL : REAL              )  ;
                   PSET  : (PVAL : SET OF SETRANGE   )  ;
                   KHAR  : (ORDCH: INTEGER           )  ;
                   STRING: (SLGTH: STRGRANGE            ;
                            SVAL : CHARSTRING        )
              END ;


TKNCLASS   = (NOTNEEDED, OPRTR, DRADRES, INDRADRES)     ;
TOKENS     = RECORD
                CLASS : SYMBOL  ;
                CASE TKNCLASS OF
                   OPRTR      : ( OP     : OPERATOR )   ;
                   DRADRES    : ( CSTADR : CSTADDR  )   ;
                   NOTNEEDED  : ( )
             END ;


IDSTRING   =        ARRAY [1..CHARMAX] OF CHAR          ;
PACKEDID   = PACKED ARRAY [1..CHARMAX] OF CHAR          ;
SETS       =        ARRAY [ 3..NSETS  ] OF
                       PACKED SET OF SETRANGE           ;
LINKS      = @ PARSESTACK                               ;
PARSESTACK = RECORD
                PTR        : SPRANGE                    ;
                PREVIOUS   : LINKS

             END                                        ;

STRUCTFORM = (SCALAR,SUBRANGE,POINTER,POWER,ARRAYS,
              RECORDS,PARAMLIST,FILES,TAGFIELD,VARIANT);
DECLKIND   = (STANDARD,DECLARED)                        ;
STP        =  @STRUCTURE                                ;
IDP        =  @IDNTFR                                   ;
STRUCTURE  =  RECORD
                 SIZE      : INTEGER   ;
                 MARKED    : BOOLEAN   ;
                 CASE FORM : STRUCTFORM OF
                    SCALAR : ( CASE SCALKIND : DECLKIND OF
                             DECLARED : (FSTCONST : IDP);
                             STANDARD : ()  ) ;
                    SUBRANGE :( RANGETYPE   : STP ;
                                MIN,MAX     : INTEGER )  ;
                    POINTER  :( ELTYPE      : STP )      ;
                    POWER    :( ELSET       : STP )      ;
                    ARRAYS   :( PACKD       : BOOLEAN    ;
                                INXTYPE,AELTYPE :STP  )  ;
                    RECORDS  :( RPACK,NOFIX: BOOLEAN     ;
                                FSTFLD      : IDP        ;
                                RECVAR      : STP )      ;
                    PARAMLIST:( ANON        : BOOLEAN    ;
                                FSTPAR      : IDP )      ;
                    FILES    :( FILTYPE     : STP )      ;
                    TAGFIELD :( TAGFIELDP   : IDP        ;
                                TAGTYPE     : STP        ;
                                FSTVAR      : STP )      ;
                    VARIANT  :( NXTVAR,SUBREC : STP      ;
                                VARVAL      : INTEGER )  ;
              END  ;

IDCLASS =
   (TYPES,KONST,VARS,FIELD,PARAMS,FUNC,PROC,PROG) ;
SETOFIDS= SET OF IDCLASS  ;
IDKIND  = (ACTUAL,FORMAL) ;
LEVRANGE = 0 .. MAXLEVEL  ;

IDNTFR = PACKED RECORD
            NAME : PACKEDID ; LLINK,RLINK : IDP ;
            IDTYPE : STP ; NEXT : IDP ;
            UNDCL : INTEGER ;
            CASE KLASS : IDCLASS OF
               KONST :   (VALUES : CONSTANT);
               VARS  :   (VKIND  : IDKIND   ;
                           VLEV   : LEVRANGE);
               PROC,FUNC :
                  (CASE PFDECLKIND : DECLKIND OF
                      STANDARD : () ;
                      DECLARED :
                         (PFLEV : LEVRANGE ;
                          PARAMPTR : STP ;
                          CASE PFKIND : IDKIND OF
                END   ;                    ACTUAL : (FORWDECL : BOOLEAN)))

        LABELSTATE
            =(REFERENCED,DEFINED,UNDEFINED) ;
        LABELPTR = @ LABELTAB ;
        FORWPFPTR= @ FORWPF   ;

        LABELTAB =
           RECORD
              LABVAL      : INTEGER   ;
              LABNIV      : LEVRANGE  ;
              NEXTLAB     : LABELPTR  ;
              STATUS      : LABELSTATE
           END ;

        FORWPF   =
           RECORD
              NAME        : PACKEDID  ;
              LINE        : INTEGER   ;
              NEXT        : FORWPFPTR
           END ;
        WHERE   = (BLCK,REC) ;
        UNIT    =
          PACKED RECORD
             NAME   : PACKEDID  ;   (* NAME OF UNIT                        *)
             FIRST  : INTEGER   ;   (* FIRST LINE OF UNIT                  *)
             ERRS   : INTEGER   ;   (* # OF ERRS BEFORE THIS UNIT          *)
             TYP    : IDCLASS   ;   (* TYPE OF UNIT                        *)
             EI04   : BOOLEAN   ;   (* ANY UNDECLARED ID'S ?               *)
             FORWP  : FORWPFPTR;    (* LIST OF FORWARDS                    *)
             FORWC  : INTEGER   ;   (* # OF FORWARDS                       *)
          END ;
VAR

           (* SCANNER VARIABLES AND DATA STRUCTURES                       *)
           (**************************************************************)

        SYMSTART : 1 .. BUFMAX ;    (* START OF CURRENT SYMBOL             *)
        I,J,K,II,JJ ,               (* COUNTERS                            *)
        CARDCNT        ,            (* CARD COUNT                          *)
        BUFLGTH        ,            (* NO OF CHARS IN INBUF                *)
        ABORTLINE      :INTEGER   ; (* START OF COMMENT OR STRING          *)
        CH             : CHAR     ; (* LAST CHAR IN INBUF                  *)
        CHCNT          ,            (* INBUF CHARACTER COUNTER             *)
        LGTH           : INTEGER  ; (* LENGTH OF LAST STRING               *)
        ACH,ACH1       : ASCII    ; (* ASCII CHARACTERS                    *)
        LCASE              : ARRAY ['A'..'Z'] OF ASCII                     ;
        TOKEN              : TOKENS                                        ;
        NAME               : PACKEDID                                      ;

        DUMMYID,IDSTR      : IDSTRING                                      ;
        EMPTYSTORE         : PACKED ARRAY [1..80] OF CHAR                  ;
        STRBUF,DUMMYSTR: CHARARRAY                                         ;

        KONSPTR            : CSTADDR                                       ;
        DIGITS,IDCHARS : SET OF CHAR                                       ;

        DEFAULT            : ARRAY [1..171] OF TOKENS                      ;
        NEXT,CHECK         : ARRAY [CHSIZE] OF INTEGER                     ;
        BASE               : ARRAY [0..171] OF INTEGER                     ;

        CHRCLASS           : ARRAY [CHAR] OF TOKENS                        ;
        INBUF              : ARRAY [1..BUFMAX] OF CHAR                     ;
        UNCLOSED           : (COMMENT, STRINGS, NONE)                      ;

           (* VARIABLES  FOR INDENTING AND DEDENTING                      *)
           (**************************************************************)

        LNEST     ,                 (* CHANGING NEST LEVELS                *)
        RNEST     : BOOLEAN     ;   (* CHANGING NEST LEVELS                *)
        NESTINCR  ,                 (* NEST INCREMENT                      *)
        NESTDECR  : INTEGER     ;   (* NEST DECREMENT                      *)
        NEST      : INTEGER     ;   (* NEST COUNTER                        *)
        RTNLEVEL  : INTEGER     ;   (* ROUTINE LEVEL COUNTER               *)


           (* STRUCTURES FOR ERROR HANDLING                               *)
           (**************************************************************)


        ERRINX    : 0 .. MAXERR ;   (* NO OF CURRENT LINE ERRORS           *)
        LASTERR   ,                 (* INDEX OF LAST ERROR                 *)
```

```
SAVEDSP  ,                        (* SAVED SP INDEX              *)
OKCOUNT  ,                        (* NUMBER OF OK FIELDS         *)
ERRPOS   : INTEGER     ;          (* ERROR POSITION              *)
ALTFLAG  ,                        (* FOR LOSTLINKS               *)
MET      : BOOLEAN     ;          (* TO FIND LOST LINKS          *)
ERRLIST  :
   ARRAY [1 .. MAXERR] OF
      PACKED RECORD
         POS    : 0..80 ; (* POSITION OF ERROR                   *)
         ERRNUM : 1..1000 (* ASSOCIATED ERROR CODE               *)
      END ;
LOSTLINKS  :
   ARRAY [1 .. 50] OF
      RECORD
         SPIND  : SPRANGE;(* POSITION OF LOST LINK               *)
         TEMP   : LINKS  ;(* ASSOCIATED LINK                     *)
      END ;


ERRDUPL             ,
EMPTYSET            : SETS                                      ;
ERRBUF              : ARRAY [1..BUFMAX]   OF CHAR               ;
ERRWARNCNT          : ARRAY [BOOLEAN] OF INTEGER                ;
MISSING             : PACKED ARRAY [SYMBOL] OF 0..1000          ;
NONTRMSGS           : PACKED ARRAY [NNONTRMS] OF 0..200         ;
ERRWARN             : ARRAY [BOOLEAN] OF
                           PACKED ARRAY [1..13] OF CHAR         ;
ABORTMES            : ARRAY [BOOLEAN] OF
                           PACKED ARRAY [1..25] OF CHAR         ;
RECOVARY            : ARRAY [1..100] OF TOKENS                  ;


    (* PARSER VARIABLES AND DATA STRUCTURES                       *)
    (**************************************************************)

STACKTOP : LINKS       ;          (*    CURRENT STACKTOP         *)
RECIND   ,                        (* RECOVARY STACK ..           *)
RIND     : INTEGER     ;          (*  .. INDEXES                 *)
FOREVER  ,                        (* LOOP CONTROL FLAG           *)
RECOVARED: BOOLEAN     ;          (* TRUE IF RECOVAR IS CALLED   *)
SAVE     ,                        (* TEMPORARY                   *)
LAST     : LINKS       ;          (* TEMPORARY                   *)
SPIND    : SPRANGE     ;          (* STABLE INDEX                *)
STABLE   :
   ARRAY [SPRANGE] OF
      PACKED RECORD
         SUC    ,                 (* SUCCESSOR FIELD             *)
         ALT    :SPRANGE ;        (* ALTERNATE FIELD             *)
         TPTYPE : TPRANGE         (* BACKWARD POINTER            *)
      END              ;
TTABLE   :
   ARRAY [TPRANGE] OF
      PACKED RECORD
         CASE TERMINAL : BOOLEAN OF
            FALSE :               (* STABLE POINTER              *)
               (POINTER : SPRANGE )                             ;
            TRUE  :
               (CLASS   : SYMBOL  )
      END              ;
    (* SEMANTIC VARIABLES AND DATA STRUCTURES                     *)
    (**************************************************************)

LSP,LSP1,LSP2,LSP3,LSP4,          (* THESE ARE STP VARIABLES     *)
SAV1,SAV2,SAV3,SAV4,LSP5,         (* USED FOR MULTI PURPOSES.    *)
FSP,FSP1,RESULT,SAV10,            (* RESULT IS USED TO KEEP      *)
SAVEDTAG,FRECVAR : STP ;          (* THE GOAL POINTER.           *)
SAV6             : STP     ;      (* TO SAVE STP POINTERS        *)
SAV5,SAV11,SAV12,                 (*                             *)
SAV7,SAV8,SAV9:  IDP       ;      (* TO SAVE IDP POINTERS        *)
LVALU,FVALU      : CONSTANT;      (* TO SAVE CONSTANTS           *)
ARRIND           ,                (* TO INDEX ARRAYST            *)
LMIN             ,                (* FOR LOWER BOUNDS            *)
LMAX             ,                (* FOR UPPER BOUNDS            *)
LCNT             ,                (* INDEX OF USER SCALARS       *)
VNO,INO          ,                (* VARIABLE COUNTERS           *)
RCASEIND         ,                (* RECORD STACK INDEX          *)
OLDS             ,                (* TO SAVE TOP VALUES          *)
RECINX           : INTEGER ;      (* TO INDEX RECST              *)

                                  (* MULTIPLE USE IDENTIFIER     *)
LCP,LCP1,LCP2,LCP3,               (* TABLE POINTERS.             *)
FCP,FCP1         : IDP ;          (*      ...BOOLEANS...         *)
                                  (* .. ALLOWED ?                *)
ALLOWDOTS        ,
EMPTY                             (* EMPTY FIELD LIST ?             *)
FORW            ;                 (* IS FORWARD ?                   *)
PRTERR          ;                 (* TO SUPPRESS ERROR MESSAGES     *)
                                  (* WHEN NECESSARY                 *)
PRNTABLE        ,                 (* SYMBOL TABLE DUMP REQUIED?     *)
KOLON           ;                 (* : MET ?                        *)
PCKD            : BOOLEAN ;       (* PACKED STRUCTURE ?             *)
LID             ,                 (* TEMPORARY IDENTIFIER           *)
PROGNAME        ,                 (* NAME FROM PROGRAM HEADING      *)
IDENTIFIER      : PACKEDID;       (* KEEPERS                        *)

                                  (*****POINTER HEADS               *)
FWPTR              : IDP       ;  (* HEAD OF CHAIN OF FORW DECL     *)
                                  (* TYPE IDS                       *)
LABPTR             ,              (* TO ALLOCATE LABEL              *)
FSTLABPTR          : LABELPTR;    (* HEAD OF LABEL CHAIN            *)
                                  (*****DCL LEVELS                  *)
OLDLEV             ,              (* TO SAVE LEVEL COUNTER          *)
LEVEL              : LEVRANGE;    (* CURRENT STATIC LEVEL           *)
TTOP,TOP,OLDTOP,
DISX               : DISPRANGE;(* TO INDEX DISPLAY                  *)
                                  (*****STATISTICS          *********)
CUNIT              : UNIT ;       (* DESCRIBES THE CURRENT UNIT     *)
STARTLINE          : INTEGER ;    (* START OF CURRENT UNIT          *)
                                  (*********************************)
INTPTR,REALPTR     ,              (*      ==STRUCTURE= =            *)
BOOLPTR,CHARPTR    ,              (*      ==TABLE      = =          *)
TEXTPTR,NILPTR     : STP  ;       (*      ==CONSTANTS= =            *)

INTID ,REALID      ,              (*      ==IDENTIFIER =            *)
TRUEID,MAXINTID    ,              (*      ==TABLE      =            *)
CHARID,BOOLID      ,              (*      ==CONSTANTS= =            *)
FALSEID,TEXTID     ,
NILID              : IDP ;
UTYPPTR         ,                 (* UNDEFINED TYPE  POINTER         *)
UVARPTR         ,                 (* UNDEFINED VAR   POINTER         *)
UKONSPTR        ,                 (* UNDEFINED KONS  POINTER         *)
UFLDPTR         ,                 (* UNDEFINED FIELD POINTER         *)
UPRCPTR         ,                 (* UNDEFINED PROC  POINTER         *)
UFCTPTR         : IDP   ;         (* UNDEFINED FUNC  POINTER         *)

FSY,FSY1              : SYMBOL ;
LFORW               : FORWPFPTR;(* TEMPORARY                        *)
OLDTOPS   : ARRAY [1..30] OF DISPRANGE ;
DISPLAY   :
    ARRAY [DISPRANGE] OF
        PACKED RECORD
            FNAME   : IDP ;
            OCCUR   : WHERE
        END ;
ARRAYST   :   ARRAY [1..20] OF STP ;
RECST     :   ARRAY [1..20] OF
                  RECORD
                      FST,LCP : IDP
                  END  ;
SIGN      : (NOSIGN,NEG,POS) ;
LKIND     : IDKIND           ;
PCASES   : ARRAY [1..20] OF
    RECORD
        FPCKD                   : BOOLEAN ;
        TAGDEFINITION           : STP     ;
        FIRSTFIELD              : IDP     ;
        VARHEAD                 ,
        INTREC                  : STP
    END ;
IDLIST   : (USERSCALAR,PFVARPAR,PFPARAM,FILEHEADS) ;
POUTINES: ARRAY [0..20] OF
              RECORD OLDLEV : LEVRANGE ;
                     OLDTOP : DISPRANGE
              END ;
```

```
                              $PAGE
(*****                                                          ******
*                                                                    *
*                     P R O C E D U R E S                            *
*                                                                    *
******                                                          *****)



   PROCEDURE STACKDUMP ;
      VAR M : LINKS ;
   BEGIN
      WRITELN ('                  **DUMP** ') ;
      M := STACKTOP ; WHILE M <> NIL DO
    BEGIN WRITELN ('          ', M @.PTR) ; M := M @. PREVIOUS END ;
      WRITELN ('          ****END DUMP ***')  END ;
   PROCEDURE INITSCAN ;
      VAR  I,J : INTEGER ;

      PROCEDURE INITRANGE (VAR  FILL : ARRAY [CHSIZE] OF INTEGER   ;
                               LOW,HIGH,VALUE : CHSIZE  )          ;
         VAR I,K,J  : CHSIZE   ;

      BEGIN
       , J         := VALUE      ;
         FOR I     := LOW TO HIGH DO
            BEGIN
               FILL [I] := J  ;
               J        := J + 1
            END
      END ; (* INITRANGE *)

   BEGIN

      ACH := ACHR (97) ;
      FOR CH := 'A' TO 'Z' DO  BEGIN
         LCASE [CH] := ACH ; ACH := SUCC (ACH) END ;
      FOR I := 0 TO 5 DO
         BEGIN
            NEXT [I] := -1 ; BASE [I] := 0 ;
            CHECK[I] := -1
         END ;

      NEXT [  6] := 68  ; NEXT [  7] := 74  ; NEXT [  8] := 99  ;
      NEXT [  9] := 65  ; NEXT [ 10] := 70  ; NEXT [ 11] := 76  ;
      NEXT [ 12] := 71  ; NEXT [ 13] := 90  ; NEXT [ 14] := 72  ;
      NEXT [ 15] := 92  ; NEXT [ 16] := 95  ; NEXT [ 17] := 94  ;
      NEXT [ 18] :=101  ; NEXT [ 19] := 64  ; NEXT [ 20] := 73  ;
      NEXT [ 21] := 77  ; NEXT [ 22] := 78  ; NEXT [ 23] := 66  ;
      NEXT [ 24] := 67  ; NEXT [ 25] := 75  ; NEXT [ 26] := 79  ;
      NEXT [ 27] := 80  ; NEXT [ 28] := 81  ; NEXT [ 29] := 82  ;
      NEXT [ 30] := 69  ; NEXT [ 31] := 84  ; NEXT [ 32] := 85  ;
      NEXT [ 33] := 86  ; NEXT [ 34] := 83  ; NEXT [ 35] := 87  ;
      NEXT [ 36] := 88  ; NEXT [ 37] := 89  ; NEXT [ 38] := 91  ;
      NEXT [ 39] := 93  ; NEXT [ 40] := 97  ; NEXT [ 41] := 98  ;
      NEXT [ 42] :=100  ; NEXT [ 43] :=103  ; NEXT [ 44] :=104  ;
      NEXT [ 45] := 96  ; NEXT [ 51] :=102  ; NEXT [ 62] :=113  ;
      NEXT [ 63] :=122  ;

      INITRANGE (NEXT,46,50,105)    ;
      INITRANGE (NEXT,52,54,110)    ;
      INITRANGE (NEXT,55,57,114)    ;
      INITRANGE (NEXT,58,61,118)    ;
      INITRANGE (NEXT,64,68,124)    ;
      INITRANGE (NEXT,69,73,130)    ;

      NEXT [ 74] :=136  ; NEXT [ 75] :=123  ; NEXT [ 76] :=171  ;
      NEXT [ 81] :=129  ; NEXT [ 94] :=155  ; NEXT [ 95] :=145  ;
      NEXT [ 96] :=157  ; NEXT [ 97] :=158  ; NEXT [ 98] :=154  ;

      INITRANGE (NEXT,77,80,137)    ;
      INITRANGE (NEXT,82,85,141)    ;
      INITRANGE (NEXT,86,93,146)    ;
      INITRANGE (NEXT,99,104,159)   ;

      NEXT [105] :=168  ; NEXT [106] :=165  ; NEXT [107] :=166  ;
      NEXT [108] :=155  ; NEXT [109] :=167  ; NEXT [110] :=169  ;
      NEXT [111] :=170  ; NEXT [112] :=135  ; NEXT [113] :=117  ;

      FOR I := 114 TO 168 DO
         BEGIN
      NEXT[I] := 0 ;
      CHECK[I] := 1 ;
    END ;

CHECK[  6] := 67 ; CHECK[  7] :=  8 ; CHECK[  8] := 98 ;
CHECK[  9] := 64 ; CHECK[ 10] :=  7 ; CHECK[ 11] := 75 ;
CHECK[ 12] := 70 ; CHECK[ 13] := 89 ; CHECK[ 14] := 71 ;
CHECK[ 15] := 91 ; CHECK[ 16] := 94 ; CHECK[ 17] := 93 ;
CHECK[ 18] :=100 ; CHECK[ 19] :=  5 ; CHECK[ 20] := 72 ;
CHECK[ 21] :=  8 ; CHECK[ 22] := 77 ; CHECK[ 23] :=  6 ;
CHECK[ 24] := 65 ; CHECK[ 25] := 74 ; CHECK[ 26] := 78 ;
CHECK[ 27] := 79 ; CHECK[ 28] :=  9 ; CHECK[ 29] := 81 ;
CHECK[ 30] := 68 ; CHECK[ 31] := 83 ; CHECK[ 32] := 84 ;
CHECK[ 33] := 85 ; CHECK[ 34] :=  9 ; CHECK[ 35] := 86 ;
CHECK[ 36] := 10 ; CHECK[ 37] := 88 ; CHECK[ 38] := 10 ;
CHECK[ 39] := 11 ; CHECK[ 40] := 95 ; CHECK[ 41] := 97 ;
CHECK[ 42] := 99 ; CHECK[ 43] :=102 ; CHECK[ 44] :=103 ;
CHECK[ 45] := 11 ; CHECK[ 50] := 12 ; CHECK[ 51] := 11 ;

INITRANGE (CHECK,46,49,104)  ;
INITRANGE (CHECK,65,68,124)  ;
INITRANGE (CHECK,69,73,129)  ;
INITRANGE (CHECK,77,79,136)  ;
INITRANGE (CHECK,82,85,140)  ;
INITRANGE (CHECK,86,88,145)  ;
INITRANGE (CHECK,99,101,158) ;

CHECK[ 52] :=109 ; CHECK[ 53] :=110 ; CHECK[ 54] := 14 ;
CHECK[ 55] := 17 ; CHECK[ 56] :=114 ; CHECK[ 57] :=115 ;
CHECK[ 58] := 18 ; CHECK[ 59] :=118 ; CHECK[ 60] := 19 ;
CHECK[ 61] :=120 ; CHECK[ 62] := 14 ; CHECK[ 63] := 20 ;
CHECK[ 64] := 21 ; CHECK[ 74] :=130 ; CHECK[ 75] := 20 ;
CHECK[ 76] :=134 ; CHECK[ 80] := 23 ; CHECK[ 81] := 21 ;
CHECK[ 89] := 24 ; CHECK[ 90] :=149 ; CHECK[ 91] := 25 ;
CHECK[ 92] :=151 ; CHECK[ 93] :=152 ; CHECK[ 94] :=155 ;
CHECK[ 95] :=140 ; CHECK[ 96] :=155 ; CHECK[ 97] := 26 ;
CHECK[ 98] := 25 ; CHECK[102] := 27 ; CHECK[103] :=162 ;
CHECK[104] := 28 ; CHECK[105] := 28 ; CHECK[106] :=164 ;
CHECK[107] :=165 ; CHECK[108] := 25 ; CHECK[109] :=166 ;
CHECK[110] :=168 ; CHECK[111] :=169 ; CHECK[112] :=171 ;
CHECK[113] :=116 ;

BASE [  6] :=  0 ; BASE [  7] :=  0 ; BASE [  8] :=  1 ;
BASE [  9] := 14 ; BASE [ 10] := 19 ; BASE [ 11] := 25 ;
BASE [ 12] := 30 ; BASE [ 13] :=  0 ; BASE [ 14] := 43 ;
BASE [ 15] :=  0 ; BASE [ 16] :=  0 ; BASE [ 17] := 49 ;
BASE [ 18] := 38 ; BASE [ 19] := 40 ; BASE [ 20] := 52 ;
BASE [ 21] := 58 ; BASE [ 22] :=  0 ; BASE [ 23] := 70 ;
BASE [ 24] := 79 ; BASE [ 25] := 73 ; BASE [ 26] := 78 ;
BASE [ 27] := 96 ; BASE [ 28] := 91 ;

FOR I := 29 TO 171 DO
    BASE [ I ] := 0 ;

BASE [ 66] :=  1 ; BASE [ 74] :=  1 ; BASE [ 75] :=  1 ;
BASE [ 77] :=  3 ; BASE [ 78] :=  2 ; BASE [ 79] :=  2 ;
BASE [ 81] :=  2 ; BASE [ 83] :=  3 ; BASE [ 84] := 13 ;
BASE [ 85] :=  8 ; BASE [ 86] := 15 ; BASE [ 88] := 13 ;
BASE [ 89] :=  3 ; BASE [ 91] :=  5 ; BASE [ 94] :=  6 ;
BASE [ 96] := 17 ; BASE [ 97] := 13 ; BASE [ 98] :=  2 ;
BASE [ 99] := 19 ; BASE [100] :=  9 ; BASE [102] := 24 ;
BASE [103] := 36 ; BASE [104] := 21 ; BASE [105] := 33 ;
BASE [106] := 28 ; BASE [107] := 30 ; BASE [109] := 27 ;
BASE [110] := 33 ; BASE [114] := 49 ; BASE [115] := 47 ;
BASE [116] := 96 ; BASE [118] := 50 ; BASE [120] := 36 ;
BASE [124] := 57 ; BASE [125] := 50 ; BASE [126] := 57 ;
BASE [127] := 59 ; BASE [129] := 49 ; BASE [130] := 62 ;
BASE [131] := 61 ; BASE [132] := 63 ; BASE [133] := 47 ;
BASE [134] := 53 ; BASE [136] := 54 ; BASE [137] := 72 ;
BASE [138] := 61 ; BASE [140] := 74 ; BASE [141] := 63 ;
BASE [142] := 61 ; BASE [143] := 75 ; BASE [145] := 76 ;
BASE [146] := 81 ; BASE [147] := 63 ; BASE [149] := 65 ;
BASE [151] := 82 ; BASE [152] := 74 ; BASE [155] := 73 ;
BASE [156] := 86 ; BASE [158] := 74 ; BASE [159] := 86 ;
BASE [160] := 84 ; BASE [162] := 80 ; BASE [164] := 92 ;
BASE [165] := 90 ; BASE [166] := 99 ; BASE [168] := 85 ;
BASE [169] := 98 ; BASE [171] :=102 ; BASE [ 72] :=  1 ;

FOR I := 1 TO 171 DO
    WITH DEFAULT [ I ] DO  CLASS := IDENT         ;

WITH DEFAULT [ 65] DO
    BEGIN  CLASS := MULOP ; OP := ANDOP  END       ;
```

```
WITH DEFAULT [ 69] DO CLASS := ARRAYSY                      ;
WITH DEFAULT [ 73] DO CLASS := BEGINSY                      ;
WITH DEFAULT [ 76] DO CLASS := CASESY                       ;
WITH DEFAULT [ 80] DO CLASS := CONSTSY                      ;

WITH DEFAULT [ 82] DO
   BEGIN  CLASS := MULOP ; OP := IDIV   END                 ;

WITH DEFAULT [ 83] DO CLASS := DOSY                         ;
WITH DEFAULT [ 87] DO CLASS := DOWNTOSY                     ;
WITH DEFAULT [ 90] DO CLASS := ELSESY                       ;
WITH DEFAULT [ 92] DO CLASS := ENDSY                        ;
WITH DEFAULT [ 95] DO CLASS := FILESY                       ;
WITH DEFAULT [ 97] DO CLASS := FORSY                        ;
WITH DEFAULT [101] DO CLASS := FORWARDSY                    ;
WITH DEFAULT [108] DO CLASS := FUNCTIONSY                   ;
WITH DEFAULT [111] DO CLASS := GOTOSY                       ;
WITH DEFAULT [112] DO CLASS := IFSY                         ;

WITH DEFAULT [113] DO
   BEGIN  CLASS := RELOP ; OP := INOP   END                 ;

WITH DEFAULT [119] DO
   BEGIN  CLASS := MULOP ; OP := IMOD   END                 ;

WITH DEFAULT [123] DO
   BEGIN  CLASS := ADDOP ; OP := OROP   END                 ;

WITH DEFAULT [117] DO CLASS := LABELSY                      ;
WITH DEFAULT [121] DO CLASS := NOTSY                        ;
WITH DEFAULT [122] DO CLASS := OFSY                         ;
WITH DEFAULT [128] DO CLASS := PACKEDSY                     ;
WITH DEFAULT [135] DO CLASS := PROCSY                       ;
WITH DEFAULT [139] DO CLASS := PROGRAMSY                    ;
WITH DEFAULT [144] DO CLASS := RECORDSY                     ;
WITH DEFAULT [148] DO CLASS := REPEATSY                     ;
WITH DEFAULT [150] DO CLASS := SETSY                        ;
WITH DEFAULT [153] DO CLASS := THENSY                       ;
WITH DEFAULT [154] DO CLASS := TOSY                         ;
WITH DEFAULT [157] DO CLASS := TYPESY                       ;
WITH DEFAULT [161] DO CLASS := UNTILSY                      ;
WITH DEFAULT [163] DO CLASS := VARSY                        ;
WITH DEFAULT [167] DO CLASS := WHILESY                      ;
WITH DEFAULT [170] DO CLASS := WITHSY                       ;

FOR I := 1 TO STRGLGTH  DO
    DUMMYSTR [I] := ' '   ;

FOR I := 1 TO CHARMAX DO
    DUMMYID [I] := ' '    ;

DIGITS := [ '0'..'9' ]         ;
IDCHARS:= DIGITS + [ 'A'..'Z' ] ;

CH := FIRSTCHAR          ;
REPEAT
   WITH CHRCLASS [CH] DO
       CLASS := OTHERSY ;
   CH := SUCC (CH)
UNTIL CH = LASTCHAR      ;
CHRCLASS [CH],CLASS := OTHERSY       ;

CHRCLASS [','].CLASS := COMMA        ;
CHRCLASS [')'].CLASS := RPARENT      ;
CHRCLASS ['.'].CLASS := PERIOD       ;
CHRCLASS ['['].CLASS := LBRACKET     ;
CHRCLASS [']'].CLASS := RBRACKET     ;
CHRCLASS ['^'].CLASS := ARROW        ;
CHRCLASS ['¬'].CLASS := ARROW        ;
CHRCLASS [';'].CLASS := SEMICOLON    ;

WITH CHRCLASS['+'] DO
   BEGIN  CLASS := ADDOP ; OP := PLUS   END ;

WITH CHRCLASS[' '] DO
   BEGIN  CLASS := ADDOP ; OP := MINUS  END ;

WITH CHRCLASS['*'] DO
   BEGIN  CLASS := MULOP ; OP := ASTR   END ;

WITH CHRCLASS['='] DO
   BEGIN  CLASS := RELOP ; OP := EQOP   END ;
```

```
    WITH CHRCLASS['/'] DO
       BEGIN  CLASS := MULOP ; OP := RDIV   END ;

    LASTERR  := 0 ;  ERRPOS  := 0       ;
    ERRINX   := 0 ;  CHCNT   := BUFMAX  ;
    CARDCNT  := 0 ;  CH      := ' '     ;
    UNCLOSED:= NONE   ;
    BUFLGTH  := 0     ;

    FOR I := 1 TO BUFMAX DO
       ERRBUF [ I ] := ' ' ;

    ERRWARN [ TRUE]  := ' *** ERROR   '   ;
    ERRWARN [FALSE]  := ' *** WARNING '   ;

    ABORTMES [ TRUE] := ' *** UNCLOSED COMMENT ***'  ;
    ABORTMES [FALSE] := ' *** UNCLOSED STRING  ***'  ;

    ERRWARNCNT [ TRUE] := 0  ;
    ERRWARNCNT [FALSE] := 0  ;

    FOR I :=3 TO NSETS DO
       EMPTYSET[ I ] := []          ;
    LNEST  := FALSE  ; RNEST := FALSE ;
    ERRDUPL          := EMPTYSET    ;
    NEST := 0 ; NESTINCR := 0 ; NESTDECR := 0
 END ; (*.INITSCAN *)
```

```
                    $PAGE
      PROCEDURE INITPARSE                 ;
        VAR  I          : INTEGER         ;
             INITIAL    : SYMBOL          ;

        PROCEDURE INITSUCC (L,H  : SPRANGE) ;
          VAR TEMP   : SPRANGE            ;
        BEGIN
          FOR TEMP   := L TO H DO
             WITH STABLE [ TEMP - 1] DO
                SUC   := TEMP
        END ;

      BEGIN

   (*****                                                               ******
   *                                                                        *
   *                       INIT GRAMMAR TABLES                              *
   *                                                                        *
   ******                                                              *****)

          FOR I := 1 TO SPSIZE DO
             WITH STABLE [ I ] DO
                ALT := FAIL        ;
   (*
         1) <PROGRAM>       ::= <PROGRAM HEADING> <BODY> .
   *)

          INITSUCC (  2,   3)                                               ;

          WITH STABLE [   1] DO    TPTYPE :=  53                            ;
          WITH STABLE [   2] DO    TPTYPE :=  55                            ;
          WITH STABLE [   3] DO
             BEGIN SUC     := OK;  TPTYPE := ORD (PERIOD        )  END ;
   (*
         2) <PROGRAM HEADING>
                            ::= PROGRAM IDENT ( <IDENT LIST> ) ;
   *)

          INITSUCC (  5,   9)                                               ;

          WITH STABLE [   4] DO    TPTYPE := ORD (PROGRAMSY     )           ;
          WITH STABLE [   5] DO    TPTYPE := ORD (IDENT         )           ;
          WITH STABLE [   6] DO    TPTYPE := ORD (LPARENT       )           ;
          WITH STABLE [   7] DO    TPTYPE :=  54                            ;
          WITH STABLE [   8] DO    TPTYPE := ORD (RPARENT       )           ;
          WITH STABLE [   9] DO
             BEGIN SUC     := OK;  TPTYPE := ORD (SEMICOLON     )  END ;
   (*
         3) <IDENT LIST> ::= IDENT / , IDENT /0/
   *)

          WITH STABLE [ 10] DO
             BEGIN SUC     := 11;  TPTYPE := ORD (IDENT         )  END ;
          WITH STABLE [ 11] DO
             BEGIN SUC     := 12;  TPTYPE := ORD (COMMA         )       ;
                   ALT     := OK
             END ;
          WITH STABLE [ 12] DO
             BEGIN SUC     := 11;  TPTYPE := ORD (IDENT         )  END ;
   (*
         4) <BODY>          ::= <DECLARATIONS> <ROUTINE DECLARATIONS>
                                <COMPOUND STATEMENT>
   *)

          INITSUCC ( 14, 15)                                                ;

          WITH STABLE [ 13] DO    TPTYPE :=  56                             ;
          WITH STABLE [ 14] DO    TPTYPE :=  71                             ;
          WITH STABLE [ 15] DO
             BEGIN SUC     := OK;  TPTYPE :=  75                       END ;
   (*
         5) <DECLARATIONS>
                            ::= <LABEL DECLARATIONS>
                                <CONSTANT DEFINITIONS>
                                <TYPE DEFINITIONS>
```

```
                               <VAR DECLARATIONS>
*)

     INITSUCC ( 17, 19)                                                   ;

     WITH STABLE [ 16] DO    TPTYPE :=  57                                ;
     WITH STABLE [ 17] DO    TPTYPE :=  58                                ;
     WITH STABLE [ 18] DO    TPTYPE :=  60                                ;
     WITH STABLE [ 19] DO
        BEGIN SUC    := OK;  TPTYPE :=  70                          END ;

(*
    6) <LABEL DECLARATIONS>
                    ::= LABEL /INTCONST/ , INTCONST/0/ : /1/
                          <EMPTY>
*)

     INITSUCC ( 21, 23)                                                   ;

     WITH STABLE [ 20] DO
        BEGIN ALT    := OK;  TPTYPE := ORD (LABELSY      )  END ;
     WITH STABLE [ 21] DO    TPTYPE := ORD (INTCONST     )          ;
     WITH STABLE [ 22] DO
        BEGIN ALT    := 24;  TPTYPE := ORD (COMMA        )  END ;
     WITH STABLE [ 23] DO
        BEGIN SUC    := 22;  TPTYPE := ORD (INTCONST     )  END ;
     WITH STABLE [ 24] DO
        BEGIN SUC    := OK;  TPTYPE := ORD (SEMICOLON    )  END ;

(*
    7) <CONSTANT DEFINITIONS>
                    ::= CONST / IDENT = <CONSTANT>/1/
                          <EMPTY>
*)

     INITSUCC ( 26, 32)                                                   ;

     WITH STABLE [ 25] DO
        BEGIN ALT    := OK;  TPTYPE := ORD (CONSTSY      )  END ;
     WITH STABLE [ 26] DO    TPTYPE := ORD (IDENT        )          ;
     WITH STABLE [ 27] DO    TPTYPE := ORD (RELOP        )          ;
     WITH STABLE [ 28] DO    TPTYPE :=  59                          ;
     WITH STABLE [ 29] DO    TPTYPE := ORD (SEMICOLON    )          ;
     WITH STABLE [ 30] DO
        BEGIN ALT    := OK;  TPTYPE := ORD (IDENT        )  END ;
     WITH STABLE [ 31] DO    TPTYPE := ORD (RELOP        )          ;
     WITH STABLE [ 32] DO
        BEGIN SUC    := 29;  TPTYPE :=  59                  END ;

(*
    8) <CONSTANT>     ::= INTCONST   REALCONST   IDENT
                          ADDOP INTCONST   ADDOP REALCONST
                          ADDOP IDENT   STRINGCONST
                          CHARCONST
*)

     WITH STABLE [ 33] DO
        BEGIN SUC    := 34;  TPTYPE := ORD (ADDOP        )          ;
              ALT    := 37
        END ;
     FOR I := 34 TO 41 DO
        WITH STABLE [I] DO   SUC    := OK                           ;
     WITH STABLE [ 34] DO
        BEGIN ALT    := 35;  TPTYPE := ORD (INTCONST     )  END ;
     WITH STABLE [ 35] DO
        BEGIN ALT    := 36;  TPTYPE := ORD (REALCONST    )  END ;
     WITH STABLE [ 36] DO    TPTYPE := ORD (IDENT        )          ;
     WITH STABLE [ 37] DO
        BEGIN ALT    := 38;  TPTYPE := ORD (IDENT        )  END ;
     WITH STABLE [ 38] DO
        BEGIN ALT    := 39;  TPTYPE := ORD (REALCONST    )  END ;
     WITH STABLE [ 39] DO
        BEGIN ALT    := 40;  TPTYPE := ORD (CHARCONST    )  END ;
     WITH STABLE [ 40] DO
        BEGIN ALT    := 41;  TPTYPE := ORD (INTCONST     )  END ;
     WITH STABLE [ 41] DO    TPTYPE := ORD (STRINGCONST)            ;

(*
    9) <TYPE DEFINITIONS>
                    ::= TYPE / IDENT = <TYPE> ; /1/    <EMPTY>
*)

     INITSUCC ( 43, 49)                                                   ;
```

```
        WITH STABLE [ 42] DO
           BEGIN ALT   := OK;  TPTYPE := ORD (TYPESY      )  END ;
        WITH STABLE [ 43] DO   TPTYPE := ORD (IDENT       )      ;
        WITH STABLE [ 44] DO   TPTYPE := ORD (RELOP       )      ;
        WITH STABLE [ 45] DO   TPTYPE :=   61                    ;
        WITH STABLE [ 46] DO   TPTYPE := ORD (SEMICOLON   )      ;
        WITH STABLE [ 47] DO
           BEGIN ALT   := OK;  TPTYPE := ORD (IDENT       )  END ;
        WITH STABLE [ 48] DO   TPTYPE := ORD (RELOP       )      ;
        WITH STABLE [ 49] DO
           BEGIN SUC   := 46;  TPTYPE :=   61                END ;

(*
      10) <TYPE >      ::= @ IDENT
                           PACKED SET OF <SIMPLE TYPE>
                                  SET OF <SIMPLE TYPE>
                           PACKED ARRAY [ <INDEX LIST> ] OF <TYPE>
                                  ARRAY [ <INDEX LIST> ] OF <TYPE>
                           PACKED RECORD <FIELD LIST> END
                                  RECORD <FIELD LIST> END
                           PACKED FILE OF <TYPE>
                                  FILE OF <TYPE>
                           <SIMPLE TYPE>
*)

        WITH STABLE [ 50] DO
           BEGIN ALT   := 52;  TPTYPE := ORD (ARROW       )      ;
                 SUC   := 51
           END ;
        WITH STABLE [ 51] DO
           BEGIN SUC   := OK;  TPTYPE := ORD (IDENT       )  END ;
        WITH STABLE [ 52] DO
           BEGIN SUC   := 53;  TPTYPE := ORD (PACKEDSY    )      ;
                 ALT   := 57
           END ;
        WITH STABLE [ 53] DO
           BEGIN SUC   := 58;  TPTYPE := ORD (ARRAYSY     )      ;
                 ALT   := 54
           END ;
        WITH STABLE [ 54] DO
           BEGIN SUC   := 64;  TPTYPE := ORD (RECORDSY    )      ;
                 ALT   := 55
           END ;
        WITH STABLE [ 55] DO
           BEGIN SUC   := 67;  TPTYPE := ORD (SETSY       )      ;
                 ALT   := 56
           END ;
        WITH STABLE [ 56] DO
           BEGIN SUC   := 70;  TPTYPE := ORD (FILESY      )  END ;

        INITSUCC ( 58, 62)                                       ;

        WITH STABLE [ 57] DO
           BEGIN ALT   := 63;  TPTYPE := ORD (ARRAYSY     )  END ;
        WITH STABLE [ 58] DO   TPTYPE := ORD (LBRACKET    )      ;
        WITH STABLE [ 59] DO   TPTYPE :=   62                    ;
        WITH STABLE [ 60] DO   TPTYPE := ORD (RBRACKET    )      ;
        WITH STABLE [ 61] DO   TPTYPE := ORD (OFSY        )      ;
        WITH STABLE [ 62] DO
           BEGIN SUC   := OK;  TPTYPE :=   61                END ;

        INITSUCC ( 64, 65)                                       ;

        WITH STABLE [ 63] DO
           BEGIN ALT   := 66;  TPTYPE := ORD (RECORDSY    )  END ;
        WITH STABLE [ 64] DO   TPTYPE :=   64                    ;
        WITH STABLE [ 65] DO
           BEGIN SUC   := OK;  TPTYPE := ORD (ENDSY       )  END ;

        INITSUCC ( 67, 68)                                       ;

        WITH STABLE [ 66] DO
           BEGIN ALT   := 69;  TPTYPE := ORD (SETSY       )  END ;
        WITH STABLE [ 67] DO   TPTYPE := ORD (OFSY        )      ;
        WITH STABLE [ 68] DO
           BEGIN SUC   := OK;  TPTYPE :=   63                END ;

        INITSUCC ( 70, 71)                                       ;

        WITH STABLE [ 69] DO
           BEGIN ALT   := 72;  TPTYPE := ORD (FILESY      )  END ;
        WITH STABLE [ 70] DO   TPTYPE := ORD (OFSY        )      ;
      WITH STABLE [ 71] DO
         BEGIN SUC    := OK;  TPTYPE :=  61                         END ;

      WITH STABLE [ 72] DO
         BEGIN SUC    := OK;  TPTYPE :=  63                         END ;

(*
    11) <INDEX LIST> ::= <SIMPLE TYPE> /, <SIMPLE TYPE>/0/
*)

      WITH STABLE [ 73] DO
         BEGIN SUC    := 74;  TPTYPE :=  63                         END ;
      WITH STABLE [ 74] DO
         BEGIN SUC    := 75;  TPTYPE := ORD (COMMA        )             ;
               ALT    := OK
         END ;
      WITH STABLE [ 75] DO
         BEGIN SUC    := 74;  TPTYPE :=  63                         END ;

(*
    12) <SIMPLE TYPE>
                     ::= ( <IDENT LIST> )    <CONSTANT>
                         <CONSTANT> .. <CONSTANT>
*)

      INITSUCC ( 77, 82)                                                ;

      WITH STABLE [ 76] DO
         BEGIN ALT    := 80;  TPTYPE := ORD (LPARENT      )     END ;
      WITH STABLE [ 77] DO    TPTYPE :=  54                             ;
      WITH STABLE [ 78] DO
         BEGIN SUC    := OK;  TPTYPE := ORD (RPARENT      )     END ;
      WITH STABLE [ 79] DO
         BEGIN SUC    := OK;  TPTYPE := ORD (IDENT        )             ;
               ALT    := 80
         END ;
      WITH STABLE [ 80] DO    TPTYPE :=  59                             ;
      WITH STABLE [ 81] DO
         BEGIN ALT    := OK;  TPTYPE := ORD (DOTDOT       )     END ;
    . WITH STABLE [ 82] DO
         BEGIN SUC    := OK;  TPTYPE :=. 59                         END ;

(*
    13) <FIELD LIST> ::= IDENT /, IDENT/0/ : <TYPE>
                         / ; <IDENT LIST> : <TYPE> /0/
                         CASE <VARIANT PART>
                         /IDENT /, IDENT/ : <TYPE> ; /
                          CASE  <VARIANT PART>
*)


      INITSUCC ( 84, 91)                                                ;

      WITH STABLE [ 83] DO
         BEGIN ALT    := 90;  TPTYPE := ORD (IDENT        )     END ;
      WITH STABLE [ 84] DO
         BEGIN ALT    := 86;  TPTYPE := ORD (COMMA        )     END ;
      WITH STABLE [ 85] DO
         BEGIN SUC    := 84;  TPTYPE := ORD (IDENT        )     END ;
      WITH STABLE [ 86] DO    TPTYPE := ORD (COLON        )             ;
      WITH STABLE [ 87] DO    TPTYPE :=  61                             ;
      WITH STABLE [ 88] DO
         BEGIN ALT    := OK;  TPTYPE := ORD (SEMICOLON    )     END ;
      WITH STABLE [ 89] DO
         BEGIN SUC    := 84;  TPTYPE := ORD (IDENT        )             ;
               ALT    := 90
         END ;
      WITH STABLE [ 90] DO    TPTYPE := ORD (CASESY       )             ;
      WITH STABLE [ 91] DO
         BEGIN SUC    := OK;  TPTYPE :=  65                         END ;

(*
    14) <VARIANT PART>
                     ::= IDENT : IDENT OF <VARIANT LIST>
                                 IDENT OF <VARIANT LIST>
*)

      INITSUCC ( 93, 96)                                                ;

      WITH STABLE [ 92] DO    TPTYPE := ORD (IDENT        )             ;
      WITH STABLE [ 93] DO
         BEGIN ALT    := 95;  TPTYPE := ORD (COLON        )     END ;
      WITH STABLE [ 94] DO    TPTYPE := ORD (IDENT        )             ;
```

```
     WITH STABLE [ 95] DO    TPTYPE := ORD (OFSY      )       ;
     WITH STABLE [ 96] DO
        BEGIN SUC   := OK;   TPTYPE :=  55                END ;
(*
   15) <VARIANT LIST>
                   ::= <VARIANT> / ; <VARIANT>/0/
*)

     INITSUCC ( 98, 99)                                       ;

     WITH STABLE [ 97] DO    TPTYPE :=  57                    ;
     WITH STABLE [ 98] DO
        BEGIN ALT   := OK;   TPTYPE := ORD (SEMICOLON )  END ;
     WITH STABLE [ 99] DO
        BEGIN SUC   := 98;   TPTYPE :=  57                END ;

(*
   16) <VARIANT>     ::= <CONSTANT LIST> : ( <FIELD PART>
*)

     INITSUCC (101,103)                                       ;

     WITH STABLE [100] DO    TPTYPE :=  59                    ;
     WITH STABLE [101] DO    TPTYPE := ORD (COLON     )       ;
     WITH STABLE [102] DO    TPTYPE := ORD (LPARENT   )       ;
     WITH STABLE [103] DO
        BEGIN SUC   := OK;   TPTYPE :=  63                END ;

(*
   17) <FIELD PART> ::= )   <FIELD LIST> )
*)

     WITH STABLE [104] DO
        BEGIN SUC   := OK;   TPTYPE := ORD (RPARENT   )       ;
              ALT   :=105
        END ;
     WITH STABLE [105] DO
        BEGIN SUC   :=106;   TPTYPE :=  54                END ;
     WITH STABLE [106] DO
        BEGIN SUC   := OK;   TPTYPE := ORD (RPARENT   )  END ;

(*
   13) <CONSTANT LIST>
                   ::= <CONSTANT> /, <CONSTANT>/0/
*)

     INITSUCC (108,109)                                       ;

     WITH STABLE [107] DO    TPTYPE :=  59                    ;
     WITH STABLE [108] DO
        BEGIN ALT   := OK;   TPTYPE := ORD (COMMA     )  END ;
     WITH STABLE [109] DO
        BEGIN SUC   :=108;   TPTYPE :=  59                END ;

(*
   19) <VAR DECLARATIONS>
                   ::= VAR /IDENT /, IDENT/0/ : <TYPE> ; /1/
                   ::= <EMPTY>
*)
     INITSUCC (111,121)                                       ;
     WITH STABLE [110] DO
        BEGIN ALT   := OK;   TPTYPE := ORD (VARSY     )  END ;
     WITH STABLE [111] DO    TPTYPE := ORD (IDENT     )       ;
     WITH STABLE [112] DO
        BEGIN ALT   :=114;   TPTYPE := ORD (COMMA     )  END ;
     WITH STABLE [113] DO
        BEGIN SUC   :=112;   TPTYPE := ORD (IDENT     )  END ;
     WITH STABLE [114] DO    TPTYPE := ORD (COLON     )       ;
     WITH STABLE [115] DO    TPTYPE :=  51                    ;
     WITH STABLE [116] DO    TPTYPE := ORD (SEMICOLON )       ;
     WITH STABLE [117] DO
        BEGIN ALT   := OK;   TPTYPE := ORD (IDENT     )  END ;
     WITH STABLE [118] DO
        BEGIN ALT   :=120;   TPTYPE := ORD (COMMA     )  END ;
     WITH STABLE [119] DO
        BEGIN SUC   :=118;   TPTYPE := ORD (IDENT     )  END ;
     WITH STABLE [120] DO    TPTYPE := ORD (COLON     )       ;
     WITH STABLE [121] DO
        BEGIN SUC   :=116;   TPTYPE :=  51                END ;

(*
  20) <ROUTINE DECLARATIONS>
```

```
                    ::= PROCEDURE IDENT
                          <FORMAL PARAMETER SECTION> ; FORWARD;
                        PROCEDURE IDENT
                          <FORMAL PARAMETER SECTION> ; <BODY> ;
                      . FUNCTION IDENT
                          <FORMAL PARAMETER SECTION> : IDENT ;
                          FORWARD ;
                        FUNCTION IDENT
                          <FORMAL PARAMETER SECTION> : IDENT ;
                          <BODY>  ;
*)
      INITSUCC (123,131)                                                    ;
      WITH STABLE [122] DO
         BEGIN ALT    :=125;   TPTYPE := ORD (PROCSY        )   END ;
      WITH STABLE [123] DO     TPTYPE := ORD (IDENT         )       ;
      WITH STABLE [124] DO
         BEGIN SUC    :=130;   TPTYPE :=  72                    END ;
      WITH STABLE [125] DO
         BEGIN ALT    := OK;   TPTYPE := ORD (FUNCTIONSY )      END ;
      WITH STABLE [126] DO     TPTYPE := ORD (IDENT         )       ;
      WITH STABLE [127] DO     TPTYPE :=  72                        ;
      WITH STABLE [128] DO     TPTYPE := ORD (COLON         )       ;
      WITH STABLE [129] DO     TPTYPE := ORD (IDENT         )       ;
      WITH STABLE [130] DO     TPTYPE := ORD (SEMICOLON     )       ;
      WITH STABLE [131] DO
         BEGIN SUC    :=133;   TPTYPE := ORD (FORWARDSY     )       ;
               ALT    :=132
         END ;
      WITH STABLE [132] DO
         BEGIN SUC    :=133;   TPTYPE :=  55                    END ;
      WITH STABLE [133] DO
         BEGIN SUC    :=122;   TPTYPE := ORD (SEMICOLON     )   END ;
(*
   21) <FORMAL PARAMETER SECTION>
                    ::= ( <FORMAL PARAMETERS> )    <EMPTY>
*)

      INITSUCC (135,136)                                                    ;

      WITH STABLE [134] DO
         BEGIN ALT    := OK;   TPTYPE := ORD (LPARENT       )   END ;
      WITH STABLE [135] DO     TPTYPE :=  73                        ;
      WITH STABLE [136] DO
         BEGIN SUC    := OK;   TPTYPE := ORD (RPARENT       )   END ;

(*
   22) <FORMAL PARAMETERS>
                    ::= <FORMAL PARAMETER> /;<FORMAL PARAMETER>/0/
*)
      INITSUCC (138,139)                                                    ;

      WITH STABLE [137] DO     TPTYPE :=  74                        ;
      WITH STABLE [138] DO
         BEGIN ALT    := OK;   TPTYPE := ORD (SEMICOLON     )   END ;
      WITH STABLE [139] DO
         BEGIN SUC    :=138;   TPTYPE :=  74                    END ;

(*
   23) <FORMAL PARAMETER>
                    ::= PROCEDURE <IDENT LIST>
                        FUNCTION <IDENT LIST> : IDENT
                        VAR  <IDENT LIST> : <TYPE>
                             <IDENT LIST> : <TYPE>

*)

      INITSUCC (141,149)                                                    ;

      WITH STABLE [140] DO
         BEGIN ALT    :=142;   TPTYPE := ORD (PROCSY        )   END ;
      WITH STABLE [141] DO
         BEGIN SUC    := OK;   TPTYPE :=  54                    END ;
      WITH STABLE [142] DO
         BEGIN ALT    :=146;   TPTYPE := ORD (FUNCTIONSY )      END ;
      WITH STABLE [143] DO     TPTYPE :=  54                        ;
      WITH STABLE [144] DO     TPTYPE := ORD (COLON         )       ;
      WITH STABLE [145] DO
         BEGIN SUC    := OK;   TPTYPE := ORD (IDENT         )   END ;
      WITH STABLE [146] DO
         BEGIN ALT    :=147;   TPTYPE := ORD (VARSY         )   END ;
      WITH STABLE [147] DO     TPTYPE :=  54                        ;
      WITH STABLE [148] DO     TPTYPE := ORD (COLON         )       ;
      WITH STABLE [149] DO
```

```
        BEGIN SUC    := OK;  TPTYPE :=  61                        END ;
(*
    24) <COMPOUND STATEMENT>
                      ::= BEGIN <STATEMENT LIST> END
*)

      INITSUCC (151,152)                                              :

      WITH STABLE [150] DO    TPTYPE := ORD (BEGINSY     )            ;
      WITH STABLE [151] DO    TPTYPE :=  76                           ;
      WITH STABLE [152] DO
        BEGIN SUC    := OK;  TPTYPE := ORD (ENDSY       )     END ;
(*
    24) <STATEMENT LIST>
                      ::= <STATEMENT> / ; <STATEMENT>/0/
*)

      INITSUCC (154,155)                                              :

      WITH STABLE [153] DO    TPTYPE :=  77                           :
      WITH STABLE [154] DO
        BEGIN ALT    := OK;  TPTYPE := ORD (SEMICOLON   )     END ;
      WITH STABLE [155] DO
        BEGIN SUC    :=154;  TPTYPE :=  77                    END ;
(*
    26) <STATEMENT>      ::= INTCONST : <UNLABELLED STATEMENT>
                          <UNLABELLED STATEMENT>
*)


      INITSUCC (157,158)                                              :

      STABLE [156].ALT                     := 158                     ;
      STABLE [156].TPTYPE                  := ORD (INTCONST    )      ;

      WITH STABLE [157] DO    TPTYPE := ORD (COLON       )            ;
      WITH STABLE [158] DO
        BEGIN SUC    := OK;  TPTYPE :=  78                    END ;
(*
    27) <UNLABELLED STATEMENT>
                      ::= BEGIN <STATEMENT LIST> END
                          IF <EXPRESSION> THEN <STATEMENT>
                          IF <EXPRESSION> THEN <STATEMENT>
                             ELSE <STATEMENT>
                          CASE <EXPRESSION> OF <CASE LIST> END
                          WHILE <EXPRESSION> DO <STATEMENT>
                          REPEAT <STATEMENT LIST> UNTIL
                             <EXPRESSION>
                          FOR IDENT := <EXPRESSION> TO
                              <EXPRESSION> DO <STATEMENT>
                          FOR IDENT := <EXPRESSION> DOWNTO
                              <EXPRESSION> DO <STATEMENT>
                          WITH <RECORD VAR LIST> DO <STATEMENT>
                          <SIMPLE STATEMENT>
*)

      INITSUCC (160,193)                                              :

      WITH STABLE [159] DO
        BEGIN ALT    :=162;  TPTYPE := ORD (BEGINSY     )     END ;
      WITH STABLE [160] DO    TPTYPE :=  76                           ;
      WITH STABLE [161] DO
        BEGIN SUC    := OK;  TPTYPE := ORD (ENDSY       )     END ;
      WITH STABLE [162] DO
        BEGIN ALT    :=168;  TPTYPE := ORD (IFSY        )     END ;
      WITH STABLE [163] DO    TPTYPE :=  84                           ;
      WITH STABLE [164] DO    TPTYPE := ORD (THENSY      )            ;
      WITH STABLE [165] DO    TPTYPE :=  77                           ;
      WITH STABLE [166] DO
        BEGIN ALT    := OK;  TPTYPE := ORD (ELSESY      )     END ;
      WITH STABLE [167] DO
        BEGIN SUC    := OK;  TPTYPE :=  77                    END ;
      WITH STABLE [168] DO
        BEGIN ALT    :=173;  TPTYPE := ORD (CASESY      )     END ;
      WITH STABLE [169] DO    TPTYPE :=  84                           ;
      WITH STABLE [170] DO    TPTYPE := ORD (OFSY        )            ;
      WITH STABLE [171] DO    TPTYPE :=  93                           ;
      WITH STABLE [172] DO
        BEGIN ALT    := OK;  TPTYPE := ORD (ENDSY       )     END ;
      WITH STABLE [173] DO
          BEGIN ALT    :=177;   TPTYPE := ORD (WHILESY     )  END ;
      WITH STABLE [174] DO      TPTYPE :=  34                     ;
      WITH STABLE [175] DO      TPTYPE := ORD (DOSY        )      ;
      WITH STABLE [176] DO
          BEGIN SUC    := OK;   TPTYPE :=  77                 END ;
      WITH STABLE [177] DO
          BEGIN ALT    :=181;   TPTYPE := ORD (REPEATSY    )  END ;
      WITH STABLE [178] DO      TPTYPE :=  75                     ;
      WITH STABLE [179] DO      TPTYPE := ORD (UNTILSY     )      ;
      WITH STABLE [180] DO
          BEGIN SUC    := OK;   TPTYPE :=  34                 END ;
      WITH STABLE [181] DO
          BEGIN ALT    :=190;   TPTYPE := ORD (FORSY       )  END ;
      WITH STABLE [182] DO      TPTYPE := ORD (IDENT       )      ;
      WITH STABLE [183] DO      TPTYPE := ORD (ASGNOP      )      ;
      WITH STABLE [184] DO      TPTYPE :=  84                     ;
      WITH STABLE [185] DO
          BEGIN SUC    :=187;   TPTYPE := ORD (TOSY        )      ;
                ALT    :=186
          END ;
      WITH STABLE [186] DO      TPTYPE := ORD (DOWNTOSY    )      ;
      WITH STABLE [187] DO      TPTYPE :=  34                     ;
      WITH STABLE [188] DO      TPTYPE := ORD (DOSY        )      ;
      WITH STABLE [189] DO
          BEGIN SUC    := OK;   TPTYPE :=  77                 END ;
      WITH STABLE [190] DO
          BEGIN ALT    :=194;   TPTYPE := ORD (WITHSY      )  END ;
      WITH STABLE [191] DO      TPTYPE :=  32                     ;
      WITH STABLE [192] DO      TPTYPE := ORD (DOSY        )      ;
      WITH STABLE [193] DO
          BEGIN SUC    := OK;   TPTYPE :=  77                 END ;
      WITH STABLE [194] DO
          BEGIN SUC    := OK;   TPTYPE :=  79                 END ;

(*
   28) <SIMPLE  STATEMENT>
                      ::= GOTO INTCONST
                          IDENT ( <EXPRESSION LIST> )
                          IDENT <VAR TAIL> := <EXPRESSION>
                          <EMPTY>
*)

      INITSUCC (196,203)                                          ;

      WITH STABLE [195] DO
          BEGIN ALT    :=197;   TPTYPE := ORD (GOTOSY      )  END ;
      WITH STABLE [196] DO
          BEGIN SUC    := OK;   TPTYPE := ORD (INTCONST    )  END ;
      WITH STABLE [197] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (IDENT       )  END ;
      WITH STABLE [198] DO
          BEGIN ALT    :=201;   TPTYPE := ORD (LPARENT     )  END ;
      WITH STABLE [199] DO      TPTYPE :=  83                     ;
      WITH STABLE [200] DO
          BEGIN SUC    := OK;   TPTYPE := ORD (RPARENT     )  END ;
      WITH STABLE [201] DO      TPTYPE :=  31                     ;
      WITH STABLE [202] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (ASGNOP      )  END ;
      WITH STABLE [203] DO
          BEGIN SUC    := OK;   TPTYPE :=  84                 END ;

(*
   29) <VARIABLE>     ::= IDENT   IDENT @   IDENT. <VARIABLE>
                          IDENT [ <EXPRESSION LIST> ]
                          IDENT [<EXPRESSION LIST>].<VARIABLE>
                          IDENT @. <VARIABLE>
*)

      INITSUCC (205,210)                                          ;

      WITH STABLE [204] DO      TPTYPE := ORD (IDENT       )      ;
      WITH STABLE [205] DO
          BEGIN SUC    :=209;   TPTYPE := ORD (ARROW       )      ;
                ALT    :=206
          END ;
      WITH STABLE [206] DO
          BEGIN ALT    :=209;   TPTYPE := ORD (LBRACKET    )  END ;
      WITH STABLE [207] DO      TPTYPE :=  33                     ;
      WITH STABLE [208] DO      TPTYPE := ORD (RBRACKET    )      ;
      WITH STABLE [209] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (PERIOD      )  END ;
      WITH STABLE [210] DO
```

```
        BEGIN SUC    := OK;   TPTYPE :=  83                              END ;
 (*
    30) <VAR TAIL>        ::= [ <EXPRESSION LIST> ]       @
                              @. <VARIABLE>      . <VARTABLE>
                              [ <EXPRESSTON LIST> ]. <VARIABLE>
                              <EMPTY>
 *)

       INITSUCC (213,216)                                                   ;

       WITH STABLE [211] DO
          BEGIN ALT    :=212;   TPTYPE := ORD (ARROW          )           ;
                SUC    :=215
          END ;
       WITH STABLE [212] DO
          BEGIN ALT    :=215;   TPTYPE := ORD (L3PACKET       )   END     ;
       WITH STABLE [213] DO     TPTYPE :=  83                             ;
       WITH STABLE [214] DO     TPTYPE := ORD (R3PACKET       )           ;
       WITH STABLE [215] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (PERIOD         )   END     ;
       WITH STABLE [216] DO
          BEGIN SUC    := OK;   TPTYPE :=  83                     END     ;
 (*
    31) <RECORD VAR LIST>
                       ::= <VARIABLE> /, <VARIABLE>/0/
 *)

       INITSUCC (218,219)                                                   ;

       WITH STABLE [217] DO     TPTYPE :=  83                             ;
       WITH STABLE [218] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (COMMA          )   END     ;
       WITH STABLE [219] DO
          BEGIN SUC    :=218;   TPTYPE :=  83                     END     ;
 (*
    32) <EXPRESSION LIST>
                       ::= <EXPRESSION> /, <EXPRESSION/0/
 *)

       INITSUCC (221,222)                                                   ;

       WITH STABLE [220] DO     TPTYPE :=  84                             ;
       WITH STABLE [221] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (COMMA          )   END     ;
       WITH STABLE [222] DO
          BEGIN SUC    :=221;   TPTYPE :=  84                     END     ;
 (*
    33) <EXPRESSION>    ::= <SIMPLE EXPRESSION> RELOP
                            <SIMPLE EXPRESSION>
 *)

       INITSUCC (224,225)                                                   ;

       WITH STABLE [223] DO     TPTYPE :=  85                             ;
       WITH STABLE [224] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (RELOP          )   END     ;
       WITH STABLE [225] DO
          BEGIN SUC    :=224;   TPTYPE :=  85                     END     ;
 (*
    34) <SIMPLE  EXPRESSION>
                       ::= ADDOP <TERM> /ADDOP <TERM>/0/
                           <TERM> /ADDOP <TERM> /0/
 *)

       INITSUCC (227,229)                                                   ;

       WITH STABLE [226] DO
          BEGIN ALT    :=227;   TPTYPE := ORD (ADDOP          )   END     ;
       WITH STABLE [227] DO     TPTYPE :=  86                             ;
       WITH STABLE [228] DO
          BEGIN ALT    := OK;   TPTYPE := ORD (ADDOP          )   END     ;
       WITH STABLE [229] DO
          BEGIN SUC    :=228;   TPTYPE :=  86                     END     ;
 (*
    35) <TERM>          ::= <FACTOR> / MULOP <FACTOR>/0/
 *)
```

```
      INITSUCC (231,232)                                                      ;
      WITH STABLE [230] DO    TPTYPE :=  87                                   ;
      WITH STABLE [231] DO
         BEGIN ALT    := OK;  TPTYPE := ORD (MULOP          )  END ;
      WITH STABLE [232] DO
         BEGIN SUC    :=231;  TPTYPE :=  87                    END ;
(*
   36) <FACTOR>         ::= INTCONST    REALCONST    STRINGCONST
                            CHARCONST    IDENT (<EXPRESSION LIST>)
                            [ <SUBSET LIST>    NOT <FACTOR>
                            ( <EXPRESSION> )   IDENT <VAR TAIL>
*)

      INITIAL    := INTCONST                      ;
      FOR    I   := 233 TO 242 DO
         WITH STABLE [ I ] DO
            BEGIN
               SUC           := OK                ;
               IF I < 237 THEN
                  BEGIN
                     ALT     := I + 1             ;
                     TPTYPE  := ORD(INITIAL)      ;
                     INITIAL:= SUCC (INITIAL)
                  END
            END    ;

      WITH STABLE [237] DO
         BEGIN SUC    :=238;  TPTYPE := ORD (NOTSY          )        ;
               ALT    :=239
         END ;
      WITH STABLE [238] DO    TPTYPE :=  87                          ;

      INITSUCC (240,241)                                             ;

      WITH STABLE [239] DO
         BEGIN ALT    :=242;  TPTYPE := ORD (LPAREN          )  END ;
      WITH STABLE [240] DO    TPTYPE :=  84                          ;
      WITH STABLE [241] DO    TPTYPE := ORD (RPAREN          )       ;

      WITH STABLE [242] DO    TPTYPE :=  92                          ;
(*
   37) <SUBSET LIST> ::= ]   <ELEMENT LIST> ]
*)

      WITH STABLE [247] DO
         BEGIN SUC    := OK;  TPTYPE := ORD (RBRACKET    )           ;
               ALT    :=248
         END ;
      WITH STABLE [248] DO
         BEGIN SUC    :=249;  TPTYPE :=  89                     END ;
      WITH STABLE [249] DO
         BEGIN SUC    := OK;  TPTYPE := ORD (RBRACKET    )      END ;
(*
   38) <ELEMENT LIST>::= <ELEMENT> /, <ELEMENT/0/
*)
      INITSUCC (251,252)                                             ;

      WITH STABLE [250] DO    TPTYPE :=  90                          ;
      WITH STABLE [251] DO
         BEGIN ALT    := OK;  TPTYPE := ORD (COMMA          )   END ;
      WITH STABLE [252] DO
         BEGIN SUC    :=251;  TPTYPE :=  90                     END ;
(*
   39) <ELEMENT>        ::= <EXPRESSION>
                            <EXPRESSION> .. <EXPRESSION>
*)

      INITSUCC (254,255)                                             ;

      WITH STABLE [253] DO    TPTYPE :=  84                          ;
      WITH STABLE [254] DO
         BEGIN ALT    := OK;  TPTYPE := ORD (DOTDOT         )   END ;
      WITH STABLE [255] DO
         BEGIN SUC    := OK;  TPTYPE :=  84                     END ;
(*
   40) <CASE LIST>      ::= <CONSTANT LIST> : <STATEMENT>
```

```
*)
      INITSUCC (257,258)
      WITH STABLE [256] DO     TPTYPE :=  59                              ;
      WITH STABLE [257] DO     TPTYPE := ORD (COLON        )              ;
      WITH STABLE [258] DO
         BEGIN SUC    := OK;   TPTYPE :=  77                          END ;
(*
   41) <EMPTY>            ::=
*)
      INITSUCC (260,264)
      WITH STABLE [259] DO                                                ;
         BEGIN ALT    :=261;   TPTYPE := ORD (LBRACKET     )          END ;
      WITH STABLE [260] DO
         BEGIN SUC    := OK;   TPTYPE :=  88                          END ;
      WITH STABLE [261] DO     TPTYPE := ORD (IDENT         )             ;
      WITH STABLE [262] DO                                                ;
         BEGIN ALT    :=265;   TPTYPE := ORD (LPARENT      )          END ;
      WITH STABLE [263] DO     TPTYPE :=  83                              ;
      WITH STABLE [264] DO                                                ;
         BEGIN SUC    := OK;   TPTYPE := ORD (RPARENT      )          END ;
      WITH STABLE [265] DO
         BEGIN SUC    := OK;   TPTYPE :=  91                          END ;
      FOR INITIAL         := IDENT TO WITHSY DO
         BEGIN
            MISSING [ INITIAL ] := 0     ;
            TTABLE [ ORD ( INITIAL ) ].CLASS := INITIAL
         END ;

      INITSUCC(267,268)                                                   ;

      WITH STABLE [266] DO     TPTYPE :=  91
      WITH STABLE [267] DO                                                ;
         BEGIN ALT    := OK;   TPTYPE := ORD (SEMICOLON    )          END ;
      WITH STABLE [268] DO
         BEGIN SUC    :=267;   TPTYPE :=  91                          END ;

      MISSING [IDENT          ] :=  2;  MISSING [INTCONST        ] := 15;
      MISSING [LPARENT        ] :=  9;  MISSING [RPARENT         ] :=  4;
      MISSING [LBRACKET       ] := 11;  MISSING [RBRACKET        ] := 12;
      MISSING [COMMA          ] := 20;  MISSING [SEMICOLON       ] := 14;
      MISSING [COLON          ] :=  5;  MISSING [ASGNOP          ] := 51;
      MISSING [BEGINSY        ] := 17;  MISSING [DOSY            ] := 54;
      MISSING [DOWNTOSY       ] := 55;  MISSING [ENDSY           ] := 13;
      MISSING [FILESY         ] := 57;  MISSING [IFSY            ] := 56;
      MISSING [TOSY           ] := 55;  MISSING [PROGRAMSY       ] :=  3;
      MISSING [THENSY         ] := 52;  MISSING [UNTILSY         ] := 53;
      MISSING [OFSY           ] :=  8;  MISSING [RELOP           ] := 16;
      MISSING [MULOP          ] := 21;  MISSING [ADDOP           ] := 22;
      MISSING [REPEATSY       ] := 23;  MISSING [LABELSY         ] := 24;
      MISSING [CONSTSY        ] := 25;  MISSING [VARSY           ] := 26;
      MISSING [TYPESY         ] := 27;  MISSING [DOTDOT          ] := 28;


      FOR I := 52 TO 93 DO
         BEGIN
            TTABLE [ I ] . TERMINAL := FALSE ;
            NONTRMSGS [ I ]          := 0
         END ;

      NONTRMSGS [63] :=  1; NONTRMSGS [61] := 10 ;
      NONTRMSGS [70] := 18; NONTRMSGS [64] := 19 ;

      NONTRMSGS [56] := 18; NONTRMSGS [57] := 18 ;
      NONTRMSGS [58] := 18; NONTRMSGS [60] := 18 ;
      NONTRMSGS [59] := 50; NONTRMSGS [87] := 58 ;
      NONTRMSGS [80] := 59;
      FOR I := 0 TO ORD (WITHSY) DO
         TTABLE [ I ].TERMINAL        := TRUE   ;

      TTABLE [52].POINTER :=   1; TTABLE [53].POINTER  :=   4;
      TTABLE [54].POINTER :=  10; TTABLE [55].POINTER  :=  13;
      TTABLE [56].POINTER :=  16; TTABLE [57].POINTER  :=  20;
      TTABLE [58].POINTER :=  25; TTABLE [59].POINTER  :=  33;
      TTABLE [60].POINTER :=  42; TTABLE [61].POINTER  :=  50;
      TTABLE [64].POINTER :=  83; TTABLE [65].POINTER  :=  92;
      TTABLE [62].POINTER :=  73; TTABLE [63].POINTER  :=  76;
      TTABLE [66].POINTER :=  97; TTABLE [67].POINTER  := 100;
      TTABLE [68].POINTER := 104; TTABLE [69].POINTER  := 107;
      TTABLE [70].POINTER := 110; TTABLE [71].POINTER  := 122;
      TTABLE [72].POINTER := 134; TTABLE [73].POINTER  := 137;
```

```
        TTABLE [74].POINTER :=140; TTABLE [75].POINTER :=150;
        TTABLE [76].POINTER :=153; TTABLE [77].POINTER :=155;
        TTABLE [78].POINTER :=159; TTABLE [79].POINTER :=195;
        TTABLE [80].POINTER :=204; TTABLE [81].POINTER :=211;
        TTABLE [82].POINTER :=217; TTABLE [83].POINTER :=220;
        TTABLE [84].POINTER :=223; TTABLE [85].POINTER :=226;
        TTABLE [86].POINTER :=230; TTABLE [87].POINTER :=233;
        TTABLE [88].POINTER :=247; TTABLE [89].POINTER :=250;
        TTABLE [90].POINTER :=253; TTABLE [91].POINTER :=256;
        TTABLE [92].POINTER :=259; TTABLE [93].POINTER :=266;

        OKCOUNT  := 0 ; RECOVARED  := FALSE    ;
        RECIND   := 0 ; LAST       := NIL      ;
        RIND     := 0 ; ERRDUPL    := EMPTYSET; ALTFLAG := FALSE ;

        FOR TOP := 0 TO DISPLIMIT DO WITH DISPLAY [TOP] DO BEGIN
           FNAME  := NIL ;
           OCCUR  := BLCK
        END ;
        IDLIST   := FILEHEADS ; OLDS     := 0   ;     PRNTABLE := FALSE ;
        TOP      := 0 ;       FWPTR     := NIL ;     PRTERR   := TRUE  ;
        SIGN     := NOSIGN; SAV10      := NIL ;     PCKD     := FALSE ;
        ARRIND   := 0 ;       FSTLABPTR:= NIL ;     ALLOWDOOTS:= FALSE ;
        LEVEL    := 0 ;       RECINX    := 0   ;     KOLON    := FALSE ;
        LKIND    := ACTUAL; RCASEIND  := 0   ;     EMPTY    := FALSE

     END  ; (* INITPARSE *)

     PROCEDURE ENTERID (LCP : IDP) ; FORWARD ;
     FUNCTION ISTRING (FSP : STP) : BOOLEAN ; FORWARD ;
     PROCEDURE GETBOUNDS(FSP : STP; VAR FMIN,FMAX : INTEGER) ; FORWARD ;
     PROCEDURE ERROR (ERRNO : INTEGER) ; FORWARD ;
     PROCEDURE ENTERSTDIDS ;
(*
     ...THIS ROUTINE INITIALIZES STANDARD NAMES AND THEIR
     ...RELATED STRUCTURE.
 *)
     BEGIN
        NEW (INTPTR, SCALAR, STANDARD) ;                 (* S *)
        WITH INTPTR @ DO                                 (* T *)
           BEGIN SIZE := 1 ; MARKED := FALSE END ;       (* P *)

        NEW (INTID, TYPES ) ;                            (* I *)
        WITH INTID @ DO BEGIN                            (* N *)
           NAME := 'INTEGER     ' ;                      (* T *)
           LLINK := NIL ; RLINK := NIL ;                 (* E *)
           IDTYPE := INTPTR ; NEXT := NIL ;              (* G *)
           UNDCL := 0                                    (* E *)
        END ;  ENTERID (INTID) ;                         (* R *)

        NEW (MAXINTID,KONST)   ;
        WITH MAXINTID @ DO  BEGIN                        (* M *)
           NAME := 'MAXINT      ' ;                      (* A *)
           LLINK := NIL ; RLINK := NIL ;                 (* X *)
           IDTYPE := INTPTR ; NEXT := NIL ;              (* I *)
           UNDCL := 0  ;
           WITH VALUES DO BEGIN                          (* T *)
              INTVAL := TRUE ; CLASS := INTGR ;
              IVAL := MAXINT
           END
        END ;  ENTERID (MAXINTID) ;

        NEW (REALPTR, SCALAR, STANDARD) ;                (* S *)
        WITH REALPTR @ DO                                (* T *)
           BEGIN SIZE := 2 ; MARKED := FALSE END;        (* P *)

        NEW (REALID,TYPES ) ;
        WITH REALID @ DO BEGIN
           NAME := 'REAL        ' ;                      (* R *)
           LLINK := NIL ; RLINK := NIL ;                 (* E *)
           IDTYPE := REALPTR ; NEXT := NIL   ;           (* A *)
           UNDCL := 0 ;                                  (* L *)
        END ; ENTERID (REALID) ;

        NEW (CHARPTR, SCALAR, STANDARD) ;
        WITH CHARPTR @ DO
           BEGIN SIZE := 1 ; MARKED := FALSE END ;

        NEW (CHARID, TYPES) ;
        WITH CHARID @ DO BEGIN                           (* C *)
           NAME := 'CHAR        ' ;                      (* H *)
           LLINK := NIL ; RLINK := NIL ;                 (* A *)
           IDTYPE := CHARPTR ; NEXT := NIL ;             (* R *)
           UNDCL := 0
        END ; ENTERID (CHARID) ;

        NEW (TRUEID, KONST) ;                            (* B *)
        NEW (FALSEID,KONST) ;                            (* O *)
        NEW (BOOLPTR,SCALAR,DECLARED) ;                  (* O *)
                                                         (* L *)
        WITH BOOLPTR @ DO  BEGIN                         (* E *)
           SIZE := 1 ; MARKED := FALSE ;                 (* A *)
           FSTCONST := TRUEID                            (* N *)
        END ;

        NEW (BOOLID, TYPES ) ;
        WITH BOOLID @ DO BEGIN
           NAME := 'BOOLEAN     ' ;
           LLINK := NIL ; RLINK := NIL ; NEXT := NIL ;
           UNDCL := 0 ; IDTYPE  := BOOLPTR
        END ;  ENTERID (BOOLID ) ;


        WITH TRUEID @ DO BEGIN
           NAME := 'TRUE        ' ;                      (* T *)
           LLINK := NIL ; RLINK := NIL ;                 (* R *)
           IDTYPE := BOOLPTR ; NEXT := NIL ;             (* U *)
           UNDCL  := 0  ;                                (* E *)
           WITH VALUES DO BEGIN
              INTVAL := TRUE ;    CLASS := INTGR ;
              IVAL := 1
           END
        END ;  ENTERID (TRUEID) ;
```

```
  WITH FALSEID@ DO BEGIN
      NAME := 'FALSE    ' ;
      LLINK := NIL ; RLINK := NIL ;                             (* F *)
      IDTYPE := BOOLPTR ; NEXT := NIL ;                         (* A *)
      UNDCL := 0 ;                                              (* L *)
      WITH VALUES DO BEGIN                                      (* S *)
         INTVAL := TRUE ;    CLASS := INTGR ;                   (* E *)
         IVAL := 0
      END
   END ;  ENTERID (FALSEID) ;

   NEW (TEXTPTR, FILES ) ;
   WITH TEXTPTR @ DO BEGIN
      SIZE := MAXINT ; MARKED := FALSE ;                        (* S *)
      FILTYPE := CHARPTR                                        (* T *)
   END ;                                                        (* P *)

   NEW (TEXTID, TYPES) ;
   WITH TEXTID @ DO BEGIN
      NAME := 'TEXT     ' ;                                     (* T *)
      LLINK := NIL ; RLINK := NIL ;                             (* E *)
      NEXT := NIL ; IDTYPE := TEXTPTR ;                         (* X *)
      UNDCL := 0                                                (* T *)
   END ; ENTERID (TEXTID) ;

   NEW (NILPTR, POINTER) ;
   WITH NILPTR @ DO BEGIN                                       (* S *)
      SIZE := 1 ; MARKED := FALSE ;                             (* T *)
      ELTYPE := NIL                                             (* P *)
   END ;

   NEW (NILID, KONST) ;
   WITH NILID @ DO BEGIN                                        (* N *)
      NAME := 'NIL      ' ;                                     (* I *)
      LLINK := NIL ; RLINK := NIL ;                             (* L *)
      NEXT := NIL ; UNDCL := 0 ;
      WITH VALUES DO BEGIN
         INTVAL := TRUE ; CLASS := INTGR ;
(********   IVAL := NILVAL       **********)
      END
   END ;  ENTERID (NILID) ;

   NEW (UTYPPTR ) ;
   WITH UTYPPTR @ DO BEGIN
      NAME := '         ' ;
      LLINK := NIL ; RLINK := NIL ; IDTYPE := NIL ;
      NEXT  := NIL ; UNDCL := 0   ; KLASS  := TYPES
   END ;

   NEW (UVARPTR ) ;
   WITH UVARPTR @ DO BEGIN
      NAME := '         ' ;
      LLINK := NIL ; RLINK := NIL ; IDTYPE := NIL ;
      NEXT  := NIL ; UNDCL := 0   ; KLASS  := VARS
   END ;

   NEW (UFLDPTR ) ;
   WITH UFLDPTR @ DO BEGIN
      NAME := '         ' ;
      LLINK := NIL ; RLINK := NIL ; IDTYPE := NIL ;
      NEXT  := NIL ; UNDCL := 0   ; KLASS  := FIELD
   END ;

   NEW (UKONSPTR) ;
   WITH UKONSPTR @ DO BEGIN
      NAME := '         ' ;
      LLINK := NIL ; RLINK := NIL ; IDTYPE := NIL ;
      NEXT  := NIL ; UNDCL := 0   ; KLASS  := TYPES ;
      WITH VALUES DO BEGIN
         INTVAL := TRUE ; CLASS := INTGR ;
         IVAL   := 0
      END
   END ;

   NEW (UPRCPTR) ;
   WITH UPRCPTR @ DO BEGIN
      NAME := '         ' ;
      LLINK := NIL ; RLINK := NIL ; IDTYPE := NIL ;
      NEXT := NIL ; UNDCL := 0 ; KLASS := PROC ;
      PFDECLKIND := DECLARED ; PFLEV := 0 ;
      PARAMPTR  := NIL ; PFKIND := ACTUAL ;
      FORWDECL := FALSE
   END ;
```

```
    NEW (UFCTPTR) ;
    WITH UFCTPTR @ DO BEGIN
       NAME := '          ' ;
       LLINK := NIL ; RLINK := NIL ; IDTYPE := NIL ;
       NEXT := NIL ; UNDCL := 0 ; KLASS := FUNC ;
       PFDECLKIND := DECLARED ; PFLEV := 0 ;
       PARAMPTR  := NIL ; PFKIND := ACTUAL ;
       FORWDECL := FALSE
    END

 END ;  (* ENTERSTDIDS *)
```

```
                 $PAGE
        PROCEDURE PRINTABLES (ALL : BOOLEAN ) ;


           VAR  I,LIM  : DISPRANGE ;
           PROCEDURE MARKS ;
(*       ...MARKS SYMBOL TABLE ENTRIES TO PREVENT
         ...MULTIPLE PRINTOUT
 *)

              VAR   I : INTEGER ;
              PROCEDURE MARKCTP (FP : IDP) ; FORWARD ;
              PROCEDURE MARKSTP (FP : STP) ;
*          ...MARK DATA STRUCTURES TO PREVENT CYCLES              *)
                 VAR  I : INTEGER ;

              BEGIN
                 IF FP <> NIL THEN
                    WITH FP @ DO
                       IF NOT MARKED THEN BEGIN
                          MARKED  := TRUE ;
                          CASE FORM OF
                             SCALAR      :           ;
                             SUBRANGE    : MARKSTP (RANGETYPE) ;
                             POINTER     : MARKSTP (ELTYPE    ) ;
                             POWER       : MARKSTP (ELSET     ) ;
                             ARRAYS      :
                                BEGIN      MARKSTP (AELTYPE   ) ;
                                           MARKSTP (INXTYPE   )  END ;
                             RECORDS     :
                                BEGIN      MARKCTP (FSTFLD    ) ;
                                           MARKSTP (RECVAR    )  END ;
                             PARAMLIST   : MARKCTP (FSTPAR    ) ;
                             FILES       : MARKSTP (FILTYPE   ) ;
                             TAGFIELD    :
                                BEGIN      MARKSTP (TAGTYPE   ) ;
                                           MARKSTP (FSTVAR    )  END ;
                             VARIANT     :
                                BEGIN      MARKSTP (NXTVAR    ) ;
                                           MARKSTP (SUBREC    )  END

                          END     (* CASE   *)
                      END         (* IF *)
              END  ;  (* MARKSTP *)


              PROCEDURE MARKCTP (FP : IDP ) ;
              BEGIN
                 IF FP <> NIL THEN
                    WITH FP @ DO  BEGIN
                       MARKCTP (LLINK) ; MARKSTP (IDTYPE) ;
                       MARKCTP (RLINK) ;
                     IF (KLASS IN [PROC,FUNC]) AND (PFDECLKIND=DECLARED)
                       THEN  MARKSTP (PARAMPTR)
                    END
              END ;  (*  MARKCTP   *)

           BEGIN     (*  MARKS      *)
              FOR I := TOP DOWNTO LIM DO MARKCTP (DISPLAY [I].FNAME)
           END     ;  (*  MARKS      *)


           PROCEDURE FOLLOWCTP (FP : IDP) ; FORWARD ;
           PROCEDURE FOLLOWSTP (FP : STP) ;
           BEGIN
              IF FP <> NIL THEN
                 WITH FP @ DO
                    IF MARKED THEN BEGIN
                       WRITELN ; WRITELN ;
                       WRITE    (' ' :20,'****STRUCTURE****') ;
                       WRITELN ;
                       MARKED := FALSE ; WRITELN ;
                       WRITE (' ' : 7,'INTERNAL STRUCTURE CODE=',FP,
                          '     SIZE=',SIZE) ;
                       WRITELN ; WRITE (' ' : 7,'FORM=') ;

                       CASE FORM OF
                          SCALAR :

                          BEGIN    WRITE ( 'SCALAR     ,') ;
                             IF SCALKIND = STANDARD THEN
                                WRITE (    'STANDARD   ')
```

```
    ELSE  WRITE ('DECLARED
       'FIRST CONSTANT ID PTR=',FSTCONST ) ;
    WRITELN
END  ;

SUBRANGE :

BEGIN  WRITE ('SUBRANGE  ,','RANGE PTR=',
    RANGETYPE,'MIN=',MIN,'MAX=',MAX) ;

    FOLLOWSTP (RANGETYPE)
END ;

POINTER  :
BEGIN
    WRITE ('POINTER    ,','ELTYPE PTR=',
    ELTYPE ) ;
    FOLLOWSTP (ELTYPE)
END ;

POWER :

BEGIN  WRITE ('SET            ,',
    'ELSET PTR =',ELSET) ;
    FOLLOWSTP (ELSET)
END ;

ARRAYS :

BEGIN
    IF PACKD THEN WRITE ('P-ARRAY  :  ,')
             ELSE WRITE ('ARRAY       ,') ;
    WRITE ('ELEMENT TYPE PTR=',AELTYPE,
          'INDEX TYPE PTR=', INXTYPE ) ;
    WRITELN ;
    FOLLOWSTP (AELTYPE ) ;
    FOLLOWSTP (INXTYPE )
END ;

RECORDS :

BEGIN
    WRITE('RECORD     ,','FIRST FIELD PTR=',
       FSTFLD,'    VAR PTR=',RECVAR ) ;
    FOLLOWCTP (FSTFLD) ;
    FOLLOWSTP (RECVAR)
END ;

PARAMLIST :

BEGIN
    WRITE    ('PARAM LIST,',
    'FIRST PARAMETER PTR=',FSTPAR) ;
    FOLLOWCTP (FSTPAR )
END ;


FILES :

BEGIN  WRITE ('FILE        ,',
    'FILE TYPE PTR=', FILTYPE ) ;
    FOLLOWSTP (FILTYPE)
END ;

TAGFIELD :

BEGIN WRITE ('TAGFIELD   ,',
    'TAGFIELD PTR=',TAGFIELDP,
    ',TAGTYPE   PTR=',TAGTYPE,
    ',FIRST VAR PTR =',FSTVAR ) ;
    FOLLOWSTP (TAGTYPE) ;FOLLOWSTP(FSTVAR)
END ;

VARIANT :

BEGIN
    WRITE     ('VARIANT    ,',
      'NEXT VARIANT PTR=',NXTVAR,
      ',SUBREC:    =',SUBREC,
      ',VARIANT VALUE=',VARVAL) ;
    FOLLOWSTP (NXTVAR ) ;
    FOLLOWSTP (SUBREC )
END
```

```
               OTHERWISE

               END  (*  CASE  *)

            END     (*  IF    *)

END ;  (* FOLLOWSTP *)

PROCEDURE FOLLOWCTP (FP : IDP) ;
BEGIN
   IF FP <> NIL THEN
      WITH FP ^ DO  BEGIN
         FOLLOWCTP (LLINK) ;
         WRITELN ; WRITELN ;
         WRITE   (' ' : 20, '****IDENTIFIER****') ;
         WRITELN ;
         WRITE (' ' : 7,'  INTERNAL CODE=',FP,
         '     NAME = ',NAME ) ;
         WRITELN ;
         WRITE (' ': 7,'LLINK=',LLINK,'  RLINK=',RLINK,
         '   NEXT=',NEXT,'  IDPTR=',IDTYPE ) ;
         WRITELN ;
         WRITE (' ' : 7,'CLASS=') ;
         CASE KLASS OF

            TYPES      : WRITE ('TYPE      ' ) ;
            KONST      :

            BEGIN
               WRITE ('CONSTANT  '). ;

               IF IDTYPE <> NIL THEN
                  IF ISTRING (IDTYPE) THEN
                     WRITE ('STRING    ',',CONST VALUE=',
                     VALUES.SVAL )   ELSE
                  IF IDTYPE =REALPTR THEN
                     WRITE ('REAL      ',',CONST VALUE=',
                     VALUES.RVAL )   ELSE
                  IF VALUES.INTVAL   THEN
                     IF IDTYPE= INTPTR THEN
                     WRITE ('INTEGER   ',',CONST VALUE=',
                     VALUES.IVAL )   ELSE
                     IF IDTYPE = BOOLPTR THEN
                     WRITE ('BOOLEAN   ',',CONST VALUE=',
                     VALUES.IVAL=1 ) ELSE
                     IF IDTYPE = CHARPTR THEN
                     WRITE ('CHARACTER ',',CONST VALUE=',
                     CHR (VALUES.IVAL) )
                     ELSE
                     WRITE (' ,CONST VALUE=',VALUES.IVAL) ;
                   WRITELN
            END ;

            VARS :

            BEGIN
               WRITE ('VARIABLE  ,') ;
               IF VKIND = ACTUAL THEN
                  WRITE ( 'ACTUAL     ,')
               ELSE  WRITE ( 'FORMAL     ,') ;
               WRITE   ('LEVEL=', VLEV : 4) ;
               IF VLEV <> 1 THEN FOLLOWCTP (NEXT)
            END ;

            FIELD : WRITELN ('FIELD      ') ;

            PARAMS : WRITELN ('PARAM LIST') ;

            PROC,FUNC :

            BEGIN
               IF KLASS = PROC THEN WRITE ('PROCEDURE ,')
               ELSE WRITE ('FUNCTION  ,') ;

               IF PFDECLKIND = STANDARD THEN
                  WRITE ('STANDARD  ,')
               ELSE  BEGIN
               IF PFKIND = ACTUAL THEN WRITE('ACTUAL     ,')
                            ELSE WRITE ('FORMAL     ,') ;
                  WRITE ('LEVEL=',PFLEV) ;
                 IF FORWDECL THEN WRITE (',FORWARD');
                  WRITE (',PARAMPTR=',PARAMPTR) ;
```

```
                     FOLLOWSTP (PARAMPTR)
                  END
               END ;
               OTHERWISE
            END ;
            FOLLOWSTP (IDTYPE ) ;
            FOLLOWCTP (RLINK  )
         END  (* WITH  *)
   END ;   (* FOLLOWCTP *)


BEGIN  (* PRINTABLES *)
   IF PRNTABLE THEN BEGIN
   WRITELN ('      **************************');
   WRITELN ('      *    SYMBOL TABLE DUMP   *');
   WRITELN ('      **************************');
   WRITELN ;

   IF ALL THEN WRITELN (' ... FOR PROGRAM...  ')
   ELSE  WRITELN (' ...INTERMEDIATE DUMP...   ') ;

   IF ALL THEN LIM := 0
   ELSE
      LIM := TOP ;


   MARKS ;
   WRITELN ;
   FOR I := TOP DOWNTO LIM DO BEGIN
      FOLLOWCTP (DISPLAY [I].FNAME) ;
      WRITELN
   END ;WRITELN ; WRITELN ; WRITELN  END
END ;  (* PRINTABLES *)
```

```
                   $PAGE
      FUNCTION COMPTYPES (FSP1,FSP2 : STP ) : BOOLEAN ;
         VAR STRICT : BOOLEAN ;

         FUNCTION COMPARE (FSP1,FSP2 : STP) : BOOLEAN ;
(*    ...COMPARE RETURNS TRUE IF FSP1 IS COMPATIBLE
      ...WITH FSP2
*)
            LABEL 1 ;
            VAR NXT1,NXT2 : IDP ; COMP : BOOLEAN ;
                MIN1,MIN2,MAX1,MAX2 : INTEGER ;

         BEGIN
            IF (FSP1=FSP2) OR (FSP1=NIL) OR (FSP2=NIL) THEN
               COMP := TRUE
            ELSE
               IF FSP1 @. FORM = FSP2 @. FORM THEN
                  CASE FSP1 @. FORM OF
                     SCALAR , (* SCALARS DECLARED ON DIFFERENT
                                 LEVELS ARE INCOMPATIBLE *)
                     RECORDS,TAGFIELD,VARIANT,FILES :
                        COMP := FALSE ;
                     SUBRANGE :
                        COMP := NOT STRICT AND
                           COMPARE (FSP1 @. RANGETYPE,FSP2@.RANGETYPE) ;
                     POINTER  :
                        COMP := (FSP1@.ELTYPE=NIL) AND
                                (FSP2@.ELTYPE=NIL) ;
                     POWER    :
                        COMP := NOT STRICT AND (FSP1@.SIZE = FSP2@.SIZE)
                           AND (FSP1 @. ELTYPE = FSP2 @. ELTYPE) ;
                     ARRAYS   :
                        BEGIN
                           IF NOT STRICT AND
                             ISTRING (FSP1) AND ISTRING (FSP2) THEN
                              BEGIN
                                 GETBOUNDS (FSP1@.INXTYPE,MIN1,MAX1) ;
                                 GETBOUNDS (FSP2@.INXTYPE,MIN2,MAX2) ;
                                 COMP := (MIN1 = 1) AND (MIN2=1)
                                    AND (MAX1=MAX2) AND
                                   (FSP1 @. PACKD = FSP2 @. PACKD) AND
                                   COMPARE (FSP1@.INXTYPE,INTPTR) AND
                                   COMPARE (FSP2@.INXTYPE,INTPTR)

                              END
                           ELSE COMP := FALSE

                        END  ;

                     PARAMLIST :
                        BEGIN
                           COMP  := TRUE ; STRICT := TRUE ;
                           NXT1 := FSP1 @. FSTPAR ;
                           NXT2 := FSP2 @. FSTPAR ;
                           WHILE COMP AND (NXT1<>NIL) AND (NXT2 <>NIL)
                              DO   BEGIN
                                  COMP := NXT1@.KLASS = NXT2@.KLASS ;
                                  IF NXT1 @. KLASS = VARS THEN
                                     COMP := COMP AND
                                        (NXT1 @. VKIND =NXT2 @.VKIND)
                                  ELSE
                                     COMP := COMP AND
                                        ((NXT1 @. PARAMPTR @. FSTPAR =
                                          NXT2 @. PARAMPTR @. FSTPAR ) AND
                                          NXT1 @. PARAMPTR @. ANON     AND
                                          NXT2 @. PARAMPTR @. ANON     AND
                                          COMPARE (NXT1 @. PARAMPTR,
                                                   NXT2 @. PARAMPTR  )) ;
                                     COMP := COMP AND
                                          COMPARE (NXT1 @. IDTYPE ,
                                                   NXT2 @. IDTYPE )      ;
                                     NXT1 := NXT1 @. NEXT  ;
                                     NXT2 := NXT2 @. NEXT  ;
                                  END
                        END ;
                     OTHERWISE
                  END
              . ELSE  (* FSP1 @. FORM <> FSP2 @. FORM  *)
                  IF FSP1 @. FORM = SUBRANGE THEN
                    COMP := NOT STRICT AND (COMPARE (FSP1@.RANGETYPE,
                                                     FSP2))

                  ELSE
                    IF FSP2 @. FORM = SUBRANGE THEN
                    COMP := NOT STRICT AND (COMPARE (FSP2@.RANGETYPE,
                                                       FSP1))
                    ELSE
                       COMP  := FALSE  ;
1  :        COMPARE := COMP ;
         END ;  (*  COMPARE *)

      BEGIN
         STRICT  := FALSE ;
         COMPTYPES := COMPARE (FSP1,FSP2 )
      END ;
```

```
                $PAGE
   PROCEDURE ENTERID (IDPTR : IDP) ;
(*
      ...ENTERS ID POINTED BY IDPTR INTO THE SYMBOL TABLE WHICH
         ON EACH DECLARATION LEVEL IS ORGANIZED AS AN UNBALANCED
         TREE
      ...
 *)

        VAR    NAM          : PACKEDID  ;
               LCP,LCP1     : IDP       ;
               LLEFT        : BOOLEAN   ;

   BEGIN
        NAM    := IDPTR @. NAME
        LCP    := DISPLAY [ TOP ].FNAME ;

        IF LCP = NIL
        THEN  DISPLAY [ TOP ].FNAME := IDPTR
        ELSE BEGIN
           REPEAT
              LCP1    := LCP ;
              IF LCP @. NAME = NAM THEN    BEGIN
                 IF PRTERR THEN ERROR (101) ;
                 LCP   := LCP @. RLINK ;
                 LLEFT := FALSE
                 END
              ELSE IF LCP @. NAME < NAM THEN
                 BEGIN
                 LCP    := LCP @. RLINK ;
                 LLEFT   := FALSE        ;
                 END
              ELSE    BEGIN
                 LCP  := LCP @. LLINK ;
                 LLEFT := TRUE
              END     (*  IF   *)

           UNTIL LCP = NIL ;

           IF LLEFT THEN   LCP1 @. LLINK := IDPTR
           ELSE            LCP1 @. RLINK := IDPTR

        END ; (* IF THEN ELSE *)

        WITH IDPTR @ DO   BEGIN
           LLINK   := NIL ;
           RLINK   := NIL ;
           UNDCL   := 0
        END

   END ; (*  ENTERID   *)


   PROCEDURE DECLARE (IDPTR : IDP ) ;
      VAR LTOP   : DISPRANGE ;
   BEGIN
      LTOP := TOP ;
      WHILE DISPLAY [TOP].OCCUR <> BLCK DO  TOP := TOP   1 ;
      PRTERR := FALSE ;
      ENTERID (IDPTR) ;
      PRTERR := TRUE ;
      IDPTR @. UNDCL := CARDCNT ;
      TOP   := LTOP
   END ;


   PROCEDURE SEARCHID (FIDCLASS : SETOFIDS; VAR FCP : IDP) ;
      LABEL 1,3   ;
      VAR    LCP   : IDP ;

   BEGIN IF RECOVARED THEN GOTO 3 ;
      FOR DISX := TOP DOWNTO 0 DO  BEGIN
         LCP := DISPLAY [DISX].FNAME ;
         WHILE LCP <> NIL DO
            IF LCP @. NAME = IDENTIFIER THEN  BEGIN
               IF LCP @. KLASS IN FIDCLASS THEN GOTO 1 ;
               IF PRTERR THEN ERROR (103) ;
               LCP := LCP @. RLINK
               END
            ELSE IF LCP @. NAME < IDENTIFIER THEN
               LCP := LCP @. RLINK
            ELSE    LCP := LCP @. LLINK
```

```
          END ; (* FOR *)

          IF PRTERR THEN  BEGIN
             ERROR (104) ;
             GUNIT.E104 := TRUE ;

       3 :   NEW (LCP) ;
             IF TYPES   IN FIDCLASS THEN LCP@     := UTYPPTR @ ELSE
             IF KONST   IN FIDCLASS THEN LCP@     := UKONSPTR @ ELSE
             IF VARS    IN FIDCLASS THEN LCP@     := UVARPTR @ ELSE
             IF FIELD   IN FIDCLASS THEN LCP@     := UFLDPTR @ ELSE
             IF PROC    IN FIDCLASS THEN LCP@     := UPRCPTR @
                                    ELSE LCP@     := UFCTPTR @ ;

             LCP @. NAME  := IDENTIFIER  ;
             DECLARE (LCP)
          END ; (*  IF  *)
 1 :
          FCP  := LCP
       END ; (* SEARCHID *)


       PROCEDURE CREATESUBRANGE(VAR FSP:STP;FMIN,FMAX : INTEGER ;
                                    FSP1 : STP) ;
       BEGIN
          NEW (FSP,SUBRANGE) ;
          WITH FSP @ DO BEGIN
             SIZE := 1 ; MARKED := FALSE ;
             RANGETYPE := FSP1 ;
             MIN := FMIN ; MAX := FMAX
          END
       END ;

       FUNCTION ISTRING (FSP : STP) : BOOLEAN ;
       BEGIN   ISTRING := FALSE ;
          IF FSP <> NIL THEN
             IF FSP @. FORM = ARRAYS THEN
                IF FSP @. ELTYPE = CHARPTR  THEN
                  ISTRING := TRUE
       END ;


       PROCEDURE SEARCHSECTION (FCP : IDP; VAR FCP1 : IDP) ;
          LABEL 16 ;

       BEGIN
          WHILE FCP <> NIL DO BEGIN
             IF FCP @. NAME = IDENTIFIER THEN GOTO 16 ;
             IF FCP @. NAME < IDENTIFIER THEN
                FCP :=  FCP @. RLINK
             ELSE
                FCP :=  FCP @. LLINK
          END ;
16 :      FCP1  := FCP
       END ;


       FUNCTION LENGTH (FSP : STP) : INTEGER ;
(*  ...ASSUMES FSP @. FORM <= SUBRANGE AND
    ...FSP <> REALPTR
*)        VAR  LMIN,LMAX   : INTEGER ;
       BEGIN
          IF FSP = NIL THEN LENGTH := 0  ELSE BEGIN
          GETBOUNDS (FSP,LMIN,LMAX) ;
          LENGTH := LMAX - LMIN + 1           END
       END ;

       PROCEDURE GETBOUNDS (FSP:STP ; VAR FMIN,FMAX : INTEGER) ;

       BEGIN
          IF (FSP=INTPTR) OR (FSP =NIL) THEN
             BEGIN   FMIN := -MAXINT ;
                     FMAX := MAXINT
             END
          ELSE
             WITH FSP@ DO  BEGIN
                IF FORM = SUBRANGE THEN
                   BEGIN  FMIN := MIN ;
                          FMAX := MAX

                   END
                ELSE  BEGIN
                   FMIN := 0 ;
```

```
               IF FSP = CHARPTR THEN FMAX := 63  ELSE
               IF FSP @. FSTCONST <> NIL
               THEN FMAX := FSP@.FSTCONST @. VALUES.IVAL
               ELSE FMAX := 0
           END      (* IF THEN ELSE *)
       END          (*     WITH     *)
END ;               (*  GETBOUNDS   *)
```

```
                         $PAGE
    (*************************************************************************
    *
    *)  PROCEDURE SEMANTICS (ACT : INTEGER) ;                                *
    *             THIS ROUTINE PERFORMS THE NECASSARY SEMANTIC              (*
    *             ACTION EACH TIME A TERMINAL HAS BEEN PARSED.               *
    *             IT ALSO GIVES ASSOCIATED ERROR MESSAGES IF THERE           *
    *             ARE ANY.                                                   *
    *                                                                        *
    *************************************************************************)
         LABEL 122,22  ;
      BEGIN
         CASE ACT OF

                    (************************)
                    (*      POINTERS        *)
                    (************************)

        -14 : PRINTABLES (LEVEL = 0 ) ;

         50 :

           BEGIN
              NEW (LSP , POINTER ) ;
              WITH LSP @ DO
                 BEGIN  SIZE := 1  ; ELTYPE := NIL  END ; ,

              RESULT  := LSP
           END ;

         51 :  (* IDENT PART OF POINTER *)

           IF NOT RECOVARED THEN BEGIN
              TTOP  := TOP ;
              WHILE DISPLAY [TTOP].OCCUR <> BLCK DO TTOP := TTOP - 1;

              SEARCHSECTION (DISPLAY [TTOP].FNAME, LCP ) ;
               IF LCP = NIL THEN
                  BEGIN
                     NEW (LCP , TYPES ) ;
                     WITH LCP @ DO  BEGIN
                        NAME   := IDENTIFIER ;
                        IDTYPE:= LSP         ;
                        NEXT   := FWPTR
                     END ;
                     FWPTR     := LCP
                  END
               ELSE  IF LCP @. KLASS <> TYPES THEN
                  ERROR (103)
               ELSE  LSP @. ELTYPE  := LCP @. IDTYPE

           END ;

                    (************************)
                    (*       ARRAYS         *)
                    (************************)


         52  :  PCKD := TRUE ;
         58  :  RESULT := NIL ;

        -73,-75 :

           BEGIN
              IF ACT = -73 THEN LSP2 := NIL ;
              NEW (LSP, ARRAYS) ;

              WITH LSP @ DO  BEGIN
                 AELTYPE  := LSP2    ;
                 INXTYPE  := NIL   ;  PACKD := PCKD ;
                 SIZE     := 1
              END ;

              LSP2  := LSP ;

              IF RESULT <> NIL THEN
                 IF RESULT @. FORM <= SUBRANGE THEN BEGIN
                    IF RESULT = REALPTR THEN         BEGIN
                       ERROR (109) ;
                       RESULT := NIL                 END
                    ELSE IF RESULT = INTPTR THEN     BEGIN
                       ERROR (149) ;
```

```
          RESULT := NIL                  END ;
        IF RESULT <> NIL THEN  BEGIN
          GETBOUNDS (RESULT,LMIN,LMAX) ;
          IF ((LMIN<MINMIN) OR (LMIN>MAXMIN)) AND
            ((LMIN<>-MAXINT) OR (LMAX<>MAXINT)) THEN
            BEGIN
              ERROR (173) ;
              WITH RESULT @ DO
                BEGIN MIN := 0 ; MAX := 0 END
            END
        END ; (*  IF  *)

        LSP @. INXTYPE := RESULT
      END   (*  IF  *)
      ELSE
       ERROR (113)


   END ;
 61 :

   BEGIN
     ARRIND  := ARRIND + 1  ;
     IF ARRIND > 20 THEN  BEGIN
       WRITELN ('**TOO MUCH NESTING OF ARRAYS **') ;
       WRITELN ('**COMPILATION ABORTED **')        ;
       GOTO 1111
     END  ;

     ARRAYST [ARRIND] := LSP2
   END ;

 62 :

   BEGIN
     LSP1 := ARRAYST [ARRIND] ;

     REPEAT
       WITH LSP1 @ DO  BEGIN
         LSP2 := AELTYPE  ;
         AELTYPE := RESULT ;

         IF INXTYPE <> NIL THEN
           SIZE := SIZE * LENGTH (INXTYPE)
       END ;

       RESULT := LSP1 ;
       LSP1 := LSP2
    UNTIL LSP1 = NIL ;
    ARRIND  := ARRIND - 1

  END ;

      (************************)
      (*        RECORDS       *)
      (************************)


63 :    (*      RECORD      *)

  BEGIN
    OLDS    := OLDS + 1 ;
    OLDTOPS [OLDS] := TOP ;
    IF TOP < DISPLIMIT THEN                  BEGIN
      TOP := TOP + 1 ;
      WITH DISPLAY [TOP] DO     BEGIN
        FNAME := NIL ;
        OCCUR := REC            END      END
    ELSE  ERROR (250) ;
    FRECVAR := NIL ; SAVEDTAG := NIL ;
    FCP   := NIL ;  LSP := NIL ;  LCP := NIL ;
    LCP1 := NIL ;  LCP2:= NIL

  END ;

83,85,89 :    (*** RECORD IDENTIFIERS***)

  BEGIN
    NEW (LCP,FIELD) ;
    WITH LCP @ DO   BEGIN
```

```
            NAME   := IDENTIFIER  ;
            IDTYPE := NIL  ;
            NEXT   := NIL
        END ;

        ENTERID (LCP) ;

        IF LCP1 = NIL THEN
            LCP1  := LCP ;
        IF LCP2 <> NIL THEN LCP2 @. NEXT := LCP ;
        LCP2    := LCP

    END ;

 86 :      (****COLON****)

    BEGIN
        IF FCP = NIL THEN FCP := LCP1 ;
        RECINX  := RECINX + 1 ;
        WITH RECST [RECINX] DO BEGIN
            LCP := LCP1 ; FST := FCP END
    END ;

 87 :

    BEGIN

        WITH RECST [RECINX] DO BEGIN
            LCP1:= LCP   ; FCP := FST END    ;
        WHILE LCP1 <> NIL DO WITH LCP1 @ DO
        BEGIN
            IDTYPE   := RESULT ;
            LCP1     := NEXT
        END ;
        RECINX   := RECINX   1 ;
        LCP1 := NIL ; LSP := NIL ;
        LCP  := NIL ; LCP2 := NIL
    END ;


  90  :  (*  CASESY   *)

    BEGIN
        NEW (LSP1,TAGFIELD) ;
        WITH LSP1 @ DO   BEGIN
            TAGFIELDP  := NIL ;   TAGTYPE := NIL ;
            FSTVAR   := NIL
        END ;
        FRECVAR    := LSP1
    END ;

  92  :

    BEGIN
        PRTERR := FALSE ;
        SEARCHID ([TYPES] , LCP1) ;
        PRTERR := TRUE  ;

        LID   := IDENTIFIER
    END ;

  93  :  (* COLON *)

    BEGIN
        NEW (LCP,FIELD) ;

        WITH LCP @ DO BEGIN
            NAME   := LID ;
            IDTYPE   := NIL ;   NEXT := NIL
        END ;
        KOLON   := TRUE ;

        ENTERID (LCP) ;
        IF FCP = NIL THEN FCP := LCP ;
        LCP2 := LCP

    END ;

  94  :          (*** <TYPE ID> ***)

    BEGIN
        SEARCHID ([TYPES],LCP1) ;
```

```
   END LSP  := LCP1 @. IDTYPE
      ;

  95   :        (***OF***)

   BEGIN
      IF KOLON THEN KOLON := FALSE
      ELSE  BEGIN
         LCP := NIL ;
         IF LCP1 = NIL THEN                       BEGIN
            IDENTIFIER := LID ;
            SEARCHID([TYPES],LCP1)                END ;
         LSP   := LCP1 @. IDTYPE
      END ;

      IF LSP <> NIL THEN  BEGIN  (* IF COLON EXISTS *)
         IF LCP <> NIL THEN LCP @. IDTYPE := LSP ;
         IF (LSP@.FORM <=SUBRANGE) OR (ISTRING(LSP)) THEN
            BEGIN
            IF LSP = REALPTR THEN ERROR (109)
            ELSE IF ISTRING(LSP) THEN ERROR (282) ;

            LSP1 @. TAGFIELDP := LCP ;
            LSP1 @. TAGTYPE   := LSP
            END
         ELSE  ERROR (110)
      END ;  (*  IF  *)
      LSP2 := NIL  ; LSP5  := NIL  ;
      LSP  := NIL  ; SAVEDTAG := LSP1

   END ;

 =96  :
   FRECVAR @. FSTVAR := LSP5 ;

 =107,_109 :

   BEGIN
      LSP1  := SAVEDTAG ;
  (* RESULT IS RETURNED BY <CONSTANT>   *)
      IF COMPTYPES(RESULT,LSP1 @. TAGTYPE) THEN       BEGIN
         LSP3 := LSP5;
         WHILE (LSP3 <> NIL) DO BEGIN
            IF LSP3 @.VARVAL = LVALU.IVAL THEN
               ERROR (158) ; (* DOUBLEVARIANT *)
            LSP3 := LSP3 @. NXTVAR
         END                                          END
      ELSE  EPROR (111) ;
      NEW (LSP3, VARIANT ) ;

      WITH LSP3 @ DO BEGIN
         NXTVAR := LSP5 ; SUBREC := LSP2 ;
         IF LVALU.INTVAL THEN VARVAL  := LVALU.IVAL
         ELSE VARVAL  := 0
      END ;

      LSP5  := LSP3 ;
      LSP2  := LSP3

   END ;

 104  :  EMPTY  := TRUE ;

 =97,=99 :

   BEGIN
      IF EMPTY THEN BEGIN
         RCASEIND := RCASEIND + 1 ;
         EMPTY := FALSE  ;
         END
      ELSE  BEGIN

      WITH RCASES[RCASEIND],INTREC @ DO BEGIN
         FSTFLD      := FCP ;
         RPACK       := PCKD  ;
         RECVAR      := FRECVAR
      END  ;
      WITH RCASES [RCASEIND] DO BEGIN
         PCKD        := FPCKD ;
         FCP         := FIRSTFIELD ;
         FRECVAR     := TAGDEFINITION ;
         LSP3        := VARHEAD ; LSP2 := INTREC
      END ;
```

```
            LSP5        := LSP3     ;
            WHILE LSP3 <> NIL DO
               WITH LSP3 @ DO BEGIN
                  LSP4   := SUBREC  ;
                  SUBREC := LSP2    ;
                  LSP3 := LSP4
            END  ;
            WITH RCASES [RCASEIND] DO
               SAVEDTAG := TAGDEFINITION   ;
            RESULT   := LSP2   ;
            RCASEIND := RCASEIND - 1

            END       (*   ELSE   *)
         END ;

      102  :


         BEGIN
            EMPTY := FALSE ;
            RCASEIND := RCASEIND + 1 ;

            WITH RCASES [RCASEIND] DO BEGIN
               FPCKD              := PCKD      ;
               TAGDEFINITION      := FRECVAR   ;
               FIRSTFIELD         := FCP       ;
               NEW (INTREC,RECORDS)            ;
               VARHEAD            := LSP3
            END ;
            FCP := NIL ; LSP := NIL ; LCP := NIL ; LCP1 := NIL ;
            LCP2 := NIL ; PCKD := FALSE ; FRECVAR := NIL
         END ;

        65   :

         BEGIN
            NEW (LSP,RECORDS) ;
            WITH LSP @ DO BEGIN
               FSTFLD := FCP ;
               RECVAR := FRECVAR ;
               RPACK  := PCKD
            END ;
            RESULT      := LSP   ;
            TOP   := OLDTOPS [OLDS] ;
            OLDS  := OLDS - 1

         END ;


      -68   :    (*  SETS  *)

         BEGIN
            IF RESULT <> NIL THEN
               IF RESULT @. FORM > SUBRANGE THEN
                  BEGIN ERROR (115) ; RESULT := NIL END
               ELSE  BEGIN
                  IF (RESULT=INTPTR) OR (RESULT=REALPTR) THEN
                     BEGIN ERROR (115) ; RESULT := NIL END
                  ELSE BEGIN
                     GETBOUNDS (RESULT,LMIN,LMAX) ;
                     IF (LMIN < 0) OR (LMAX > 72) THEN
                        BEGIN ERROR (169) ; RESULT := NIL END
                  END
               END ;

            NEW (LSP , POWER ) ;
            LSP @. ELSET := RESULT ;
            RESULT := LSP

         END ;
      -71  :    (* FILESY *)

         BEGIN
            NEW (LSP,FILES )   ;

            WITH LSP @ DO BEGIN
               SIZE  := MAXINT ;
               FILTYPE := RESULT
            END
```

```
     END ;

                            (*******************)
                            (*  SIMPLE TYPE    *)
                            (*******************)

 76  :     (* USER DEFINED SCALARS *)

     BEGIN
        TTOP  := TOP ;
        WHILE DISPLAY [TOP].OCCUR <> BLCK DO TOP := TOP - 1 ;
        NEW (RESULT,SCALAR,DECLARED) ;
        WITH RESULT @ DO SIZE := 1 ;

        LCP1 := NIL ; LCNT := 0 ;
        IDLIST  . := USERSCALAR  ; LSP := RESULT
     END ;

 78  :

     BEGIN

        WITH RESULT @ DO FSTCONST := LCP1 ;
        TOP  := TTOP
     END ;

 33 :

     BEGIN   IF NOT RECOVARED THEN
        IF NOT (TOKEN.OP IN [PLUS,MINUS])    THEN
           ERROR (6)
        ELSE IF TOKEN.OP = PLUS THEN SIGN := POS
        ELSE SIGN := NEG
     END ;

 34,40 :  (*  INTEGER CONSTANT *)

     IF NOT RECOVARED THEN BEGIN
        IF SIGN = NEG THEN
           WITH TOKEN.CSTADR @ DO
              BEGIN INTVAL := TRUE ; IVAL := - IVAL END ;
        RESULT  := INTPTR ;
        LVALU := TOKEN.CSTADR @ ;
        SIGN := NOSIGN ; ALLOWDOTS := TRUE
     END  ELSE RESULT := NIL ;

 35,38 :

     IF NOT RECOVARED THEN BEGIN
        IF SIGN = NEG THEN
           WITH TOKEN.CSTADR @ DO
              BEGIN INTVAL := FALSE; RVAL := - RVAL END ;
        RESULT := REALPTR ;
        LVALU  := TOKEN.CSTADR @ ;
        SIGN   := NOSIGN
     END ELSE RESULT := NIL ;

 36,37 :  (* IDENT *)

     IF NOT RECOVARED THEN  BEGIN
        ALLOWDOTS := FALSE  ;
        SEARCHID ([TYPES,KONST], LCP ) ;

        IF LCP @, KLASS = KONST THEN
           BEGIN
              ALLOWDOTS := TRUE ;
              RESULT := LCP @. IDTYPE ;
              IF SIGN <> NOSIGN THEN
                 IF SIGN = NEG  THEN
                 WITH LCP @, LCP @. VALUES DO
                    IF CLASS = INTGR THEN IVAL := -IVAL ELSE
                    IF CLASS = REEL  THEN RVAL := -RVAL
                    ELSE ERROR (105)   ;

                 SIGN := NOSIGN  ;
              LVALU := LCP @. VALUES
           END
        ELSE
           BEGIN
              RESULT := LCP @. IDTYPE ;
              IF SIGN <> NOSIGN THEN ERROR (105)
```

```
          END
    END   ELSE RESULT := NIL ;
41,39  :  (* STRING CONSTANTS *)
    IF NOT RECOVARED THEN BEGIN
       LSP1 := CHARPTR     ;
       IF LGTH = 1 THEN
          BEGIN RESULT := LSP1 ; ALLOWDOTS := TRUE END
       ELSE BEGIN
          NEW (RESULT,ARRAYS) ;
          WITH RESULT @ DO BEGIN
             AELTYPE := LSP1 ; PACKD := TRUE ;
             SIZE := LGTH    ; LSP1  := NIL ;
             CREATESUBRANGE (LSP1,1,LGTH,INTPTR) ;
             INXTYPE := LSP1
          END
       END ;
       LVALU := TOKEN.CSTADR @
    END  ELSE RESULT := NIL ;
81   : IF (NOT ALLOWDOTS) AND (NOT RECOVARED) THEN
          ERROR (6)
      ELSE
    BEGIN
       IF RESULT <> NIL THEN BEGIN
          IF RESULT @. FORM = SUBRANGE THEN
             RESULT := RESULT @. RANGETYPE    ;
          IF RESULT @. FORM <> SCALAR THEN
             BEGIN ERROR (148) ; RESULT := NIL END
          ELSE IF (NOT LVALU.INTVAL) AND (RESULT <> REALPTR)
               THEN  BEGIN
                  ERROR( 29) ; RESULT := NIL
               END
       END ;
       SAV10:= RESULT ;
       IF NOT RECOVARED THEN FVALU := LVALU
    END ;
82 :
    IF NOT ALLOWDOTS THEN ERROR (6) ELSE BEGIN
       ALLOWDOTS := FALSE ;
       IF RESULT <> NIL THEN BEGIN
          IF RESULT @. FORM = SUBRANGE THEN
             RESULT := RESULT @. RANGETYPE    ;
          IF RESULT @. FORM <> SCALAR THEN
             BEGIN ERROR (148) ; RESULT := NIL END
          ELSE IF (NOT LVALU.INTVAL) AND (RESULT <> REALPTR)
               THEN  BEGIN
                  ERROR( 29) ; RESULT := NIL
               END
       END ;
       FSP1  := SAV10;
       IF (FSP1=REALPTR) AND (RESULT=REALPTR) THEN
          BEGIN ERROR (30) ;
                RESULT := NIL
          END    ;
       IF (FSP1=RESULT) AND (RESULT <> NIL) THEN
          CREATESUBRANGE (LSP,FVALU.IVAL,LVALU.IVAL,FSP1)
       ELSE
          BEGIN
             CREATESUBRANGE(LSP,-MAXINT,MAXINT,FSP1) ;
             IF (FSP1 <> NIL) AND (RESULT <> NIL) THEN
                ERROR (107)
          END ;
       RESULT := LSP
    END ;
```

```
                    (*********************)
                    (* VAR DECLARATIONS *)
                    (*********************)
```

```
110 :  (***VAR***)

    BEGIN      VNO := 0 ; INO := 0 ;
               LCP1 := NIL ; LCP2 := NIL
    END ;

111,113,117,119 :  (* IDENT *)

    IF NOT RECOVARED THEN  BEGIN
       NEW (LCP , VARS) ;
       VNO := VNO + 1    ;

       WITH LCP @ DO BEGIN
          NAME := IDENTIFIER ; NEXT :=NIL ;
          IDTYPE := NIL ;  VKIND := ACTUAL ;
          VLEV := LEVEL
       END ;

       ENTERID (LCP) ;

       IF LCP1 = NIL THEN LCP1 := LCP ;
       IF LCP2 <> NIL THEN LCP2 @. NEXT := LCP ;
       LCP2 := LCP

    END ;

114,120  : (* COLON *)
   SAV11 := LCP1 ;

121,115        :

    BEGIN
       LCP1  := SAV11 ;
       WHILE (LCP1 <> NIL) DO
         WITH LCP1 @ DO BEGIN
              IDTYPE := RESULT ;
              LCP1 := NEXT
       END

    END ;


                        (*********************)
                        (* TYPE DEFINITIONS  *)
                        (*********************)


43,47      :  (* TYPE DEFINITIONS *)

    IF NOT RECOVARED THEN  BEGIN
       NEW (LCP,TYPES) ;
       WITH LCP @ DO BEGIN
          NAME := IDENTIFIER ; IDTYPE := NIL ; NEXT := NIL
       END  ; SAV11:= LCP
    END  ELSE SAV11 := NIL ;

44,48 : IF TOKEN.OP <> EQOP  THEN ERROR (16) ;
45,49 :

    IF SAV11 <> NIL THEN BEGIN
       LCP  := SAV11  ;
       ENTERID (LCP) ;
       LCP @. IDTYPE := RESULT ;
       LCP  := NIL    ;

    ...HAS ANY FORWARD REFERENCE NOW
    ...BEEN SATISFIED  ?

       LCP1 := FWPTR ;
       WHILE (LCP1 <> NIL) DO BEGIN
          IF LCP1 @. NAME = LCP @. NAME THEN BEGIN
             LCP1 @. IDTYPE @. ELTYPE := RESULT  ;
             IF LCP1 <> FWPTR THEN
                LCP2 @.NEXT := LCP1 @. NEXT
             ELSE
                FWPTR := LCP1 @. NEXT
          END ;  (*  IF   *)

          LCP2  := LCP1 ;
          LCP1 := LCP1 @. NEXT
       END  ;  (* WHILE *)
```

```
            END ;

                            (************************)
                            (* CONSTANT DEFINITIONS *)
                            (************************)


        26,30 :

            IF NOT RECOVARED THEN BEGIN
               NEW (LCP,KONST) ;
               WITH LCP @ DO BEGIN
                  NAME := IDENTIFIER ;
                  IDTYPE := NIL ; NEXT := NIL
               END ;  SAV11 := LCP
            END ELSE SAV11  := NIL ;

        27,31 :
            IF TOKEN.OP <> EQOP  THEN ERROR (15) ; .

        28,32 :

            IF SAV11 <> NIL THEN BEGIN
               LCP := SAV11 ;
               ENTERID (LCP) ;

               WITH LCP @ DO BEGIN
                  IDTYPE := RESULT ;
                  VALUES := LVALU
               END  ;  LCP  := NIL
            END ;


                            (**********************)
                            (* LABEL DECLARATIONS *)
                            (**********************)


        21,23 :  (***LABEL***)

            BEGIN
               LABPTR := FSTLABPTR ;
               WHILE LABPTR <> NIL DO
                  WITH LABPTR @ DO BEGIN
                     IF LABNIV < LEVEL THEN GOTO 22 ;
                     IF LABVAL = TOKEN.CSTADR @. IVAL THEN
                        BEGIN  ERROR (165) ;
                               GOTO 122
                        END
                     ELSE  LABPTR := NEXTLAB
               END ;
 22 :
               NEW (LABPTR) ;
               WITH LABPTR @ DO  BEGIN
                  LABVAL := TOKEN.CSTADR @. IVAL ;
                  LABNIV := LEVEL      ;
                  STATUS := DEFINED ;
                  NEXTLAB:= FSTLABPTR
               END ;

               FSTLABPTR := LABPTR    ;
      122 :
            END ;


                            (************************)
                            (* ROUTINE DECLARATIONS *)
                            (************************)


        122 : FSY := PROCSY            ;
        125 : FSY := FUNCTIONSY        ;

        123,126 :

            BEGIN
               SEARCHSECTION (DISPLAY [TOP].FNAME ,  LCP) ;
(*
            ***DECIDE WHETHER IT HAS BEEN
            ***DECLARED FORWARD ?
*)
```

```
                FORW := FORW AND (FSY = PROCSY)
              ELSE IF KLASS = FUNC
              THEN FORW := FORW AND (FSY = FUNCTIONSY)
              ELSE  FORW := FALSE ;

    (*******   IF FORW THEN
                WITH GUNIT DO BEGIN
                  FORWC := FORWC + 1 ;
                  LFORW := FORWP     ;
                  WHILE LFORW @. NAME <> IDENTIFIER DO
                    LFORW := LFORW @. NEXT ;
                  LFORW @. LINE := 0
                END      ****************)
              ELSE  ERROR (160) (* MULTIPLE ROUTINE ID *)
            END
          ELSE  FORW := FALSE  ;

          IF FORW THEN LSP := LCP @. PARAMPTR
            (* TAKE PREVIOUSLY DEFINED PARAMETERS *)
          ELSE BEGIN
            IF FSY = PROCSY THEN
              NEW (LCP,PROC,DECLARED,ACTUAL)
            ELSE
              NEW (LCP,FUNC,DECLARED,ACTUAL)    ;

            NEW (LSP, PARAMLIST ) ;
            WITH LSP @ DO BEGIN
              SIZE := 0 ; MARKED := FALSE ;
              ANON := TRUE ; FSTPAR := NIL
            END ;


            WITH LCP @ DO BEGIN
              NAME := IDENTIFIER ;
              PARAMPTR := LSP ; (* SET PARAMETER HEAD *)
              IDTYPE := NIL ; NEXT := NIL ;

              PFLEV := LEVEL ; FORWDECL := FALSE

            END ;

            ENTERID (LCP) ;
          END ;
(**************
          IF RECOVARED THEN
            IF FSY = PROCSY THEN LCP := UPRCPTR
                          ELSE LCP := UFCTPTR ;

**************)

          IF LEVEL < MAXLEVEL THEN
            LEVEL := LEVEL + 1  (* INCR ROUTINE LEVEL *)
          ELSE ERROR (251) ;
          WITH ROUTINES [LEVEL] DO BEGIN
            OLDLEV   := LEVEL ;
            OLDTOP   := TOP
          END ;


          IF TOP < DISPLIMIT THEN
            BEGIN  TOP := TOP + 1 ;
              WITH DISPLAY [TOP] DO BEGIN
                OCCUR := BLCK ;  (* START OF A NEW BLOCK *)
                IF FORW THEN
                  FNAME := LCP @. NEXT
                ELSE FNAME := NIL
              END
            END
          ELSE ERROR (250) ;

          SAV5  := LCP  ; (* SAV5 KEEPS IDP POINTER *)
          SAV6  := LSP    (* SAV6 KEEPS STP POINTER *)

        END ;

     124, 127 :

        BEGIN
          LCP := SAV5 ;
```

```
         LCP1 := SAV6 @. FSTPAR        ;

         WHILE LCP1 <> NIL DO
            WITH LCP1 @ DO BEGIN
               IF KLASS = VARS THEN  VLEV := LEVEL
               ELSE PFLEV := LEVEL  ;

               ENTERID (LCP1 ) ;
               LCP1  := NEXT
            END
      END ;

  129 :  (*  FUNCTION TYPE *)

      BEGIN
         SEARCHID ([TYPES], LCP1) ;
         LSP := LCP1 @. IDTYPE ;
         LCP @. IDTYPE := LSP  ;

         IF LSP <> NIL THEN
            IF LSP @. FORM >= POWER THEN BEGIN
               ERROR (120) ;
               LCP @. IDTYPE := NIL
            END
      END ;

  131  :  (*  FORWARDID *)

         IF LCP @. FORWDECL THEN
            ERROR (161)
         ELSE BEGIN
            LCP @. FORWDECL := TRUE ;
            GUNIT.FORWC := GUNIT.FORWC + 1 ;
            NEW (LFORW) ;

            WITH LFORW @ DO BEGIN
               NAME := LCP @. NAME ;
               LINE := CARDCNT ;
               NEXT := GUNIT.FORWP ;
               GUNIT.FORWP := LFORW
            END
         END ;

  133  :
     BEGIN
        LCP1 := SAV6 @. FSTPAR ;
        WHILE LCP1 <> NIL DO
           WITH LCP1 @ DO BEGIN
              LLINK := NIL ; RLINK := NIL ;
              LCP1 := NEXT
        END ;

        WITH ROUTINES [LEVEL] DO BEGIN
           LEVEL  := OLDLEV ;
           TOP    := OLDTOP
        END
     END ;


  134 :
     BEGIN
        IF FORW THEN ERROR (119) ;
        LCP1 := NIL ; LCP2 := NIL ;
        IDLIST := PFVARPAR ;
        LKIND := ACTUAL ; SAV9 := NIL
     END ;

  138 :
     BEGIN
        IF SAV9 = NIL THEN
           SAV6 @. FSTPAR := SAV7
        ELSE
           SAV9 @. NEXT   := SAV7 ;
        SAV9 := SAV8
     END ;

  146    :  LKIND  := FORMAL ;

 =149    :
    BEGIN        (* SAV7 : FIRST ID IN THE SECTION *)
                 (* SAV8 : LAST  ID IN THE SECTION *)
       LCP1  := SAV7;
```

```
        WHILE LCP1 <> NIL DO
           WITH LCP1 @ DO BEGIN
           IDTYPE    := RESULT  ;
           LCP1      := NEXT
        END
    END ;
140  : BEGIN FSY1  := PROCSY   ; IDLIST := PFPARAM END ;
142  : BEGIN FSY1  := FUNCTIONSY ; IDLIST := PFPARAM END ;

145  :    (********<TYPE ID>******)
    BEGIN
       SEARCHID ([TYPES],LCP1) ;
       LSP  := LCP1 @. IDTYPE  ;
       IF LSP <> NIL THEN
          IF NOT (LSP@.FORM IN [SCALAR,SUBRANGE,POINTER]) THEN
             BEGIN   ERROR (120)   ;
                     LSP    := NIL
             END ;

       LCP  := SAV7 ;
       IF LSP <> NIL THEN
          WHILE LCP <> NIL DO
             WITH LCP @ DO BEGIN
               IDTYPE  := LSP ;
               LCP   := NEXT
          END
    END ;

10,12  :

       CASE IDLIST OF
          PFPARAM  : (* INITIALY LCP1 = NIL *)
             BEGIN
                IF FSY1 = PROCSY THEN
                   NEW (LCP,PROC,DECLARED,FORMAL)
                ELSE
                   NEW (LCP,FUNC,DECLARED,FORMAL) ;
                WITH LCP @ DO BEGIN
                NAME := IDENTIFIER ; IDTYPE := NIL;UNDCL:=0;
                LLINK := NIL ; RLINK := NIL ; NEXT := NIL ;

                PFLEV := 0
                END  ;

                LCP @. PARAMPTR := NIL ;
                IF LCP1 = NIL THEN
                   BEGIN  SAV7:= LCP ;
                          LCP1 := LCP
                   END
                ELSE  LCP1 @. NEXT  := LCP ;

                LCP1  := LCP ;
                SAV8  := LCP
             END ;

         PFVARPAR :
            BEGIN

                NEW (LCP,VARS) ;
                WITH LCP @ DO BEGIN
                  NAME := IDENTIFIER ; IDTYPE := NIL ;
                  UNDCL := 0 ; LLINK := NIL ; RLINK := NIL ;
                  NEXT := NIL ; VLEV := 0 ; VKIND := LKIND
                 END  ;


                IF LCP1 = NIL THEN
                   BEGIN  SAV7 := LCP ;
                          LCP1 := LCP
                   END
                ELSE LCP1 @. NEXT := LCP ;

                LCP1  := LCP ;
                SAV8  := LCP1
             END ;


         FILEHEADS  :         ;


         USERSCALAR :     (* INITIALY LCP1 = NIL *)
            IF NOT RECOVARED THEN BEGIN
               NEW (LCP,KONST) ;
```

```
                     NAME := IDENTIFIER ; IDTYPE := LSP ;
                     NEXT := LCP1 ;
                     WITH VALUES DO BEGIN
                        INTVAL := TRUE ;    CLASS := INTGR ;
                        IVAL   := LCNT
                     END
                 END   ;
                 ENTERID (LCP) ;
                 LCNT := LCNT + 1 ;
                 LCP1 := LCP
              END
         END  ;
      OTHERWISE  (* NO ACTION  *)
   END
END ;    (*  SEMANTICS  *)
```

```
              $PAGE
    PROCEDURE MARKSET (VAR CORRSET :SETS;I :INTEGER); FORWARD ;

 (********************************************************************
 *                                                                   *
 *)  PROCEDURE ERROR  (ERRNO : INTEGER) ;                           (*
 *              THIS PROCEDURE COLLECTS ERROR AND WARNING CODES      *
 *              TO BE PRINTED AFTER EACH LINE.                       *
 *                                                                   *
 ********************************************************************)

        VAR ERRORWARN : BOOLEAN ;
    BEGIN

        ERRORWARN      := ERRNO > 500                    ;
        ERRWARNCNT [ERRORWARN]
                       := ERRWARNCNT [ERRORWARN] + 1 ;
        IF ERRWARNCNT [FALSE] > 70 THEN
           BEGIN
              WRITELN ('***TOO MUCH SEVERE ERRORS',
                       '... COMPILATION ABORTED***') ;
              GOTO 111
           END ;


        IF ERRINX = MAXERR
        THEN ERRNO  := 255
        ELSE ERRINX := ERRINX + 1  ;

        IF SYMSTART > LASTERR THEN
           BEGIN
              LASTERR := SYMSTART  ;
              ERRPOS  := ERRPOS + 1 ;
              IF ERRPOS >= 10
              THEN
                 ERRBUF [SYMSTART] := '*'
              ELSE
                 ERRBUF [SYMSTART] := CHR (ERRPOS + ORD ('0'))
           END ;

        WITH ERRLIST [ERRINX] DO
           BEGIN
              ERRNUM := ERRNO  ;
              POS    := ERRPOS
           END ;

        MARKSET (ERRDUPL, ERRNO)
    END ; (* ERROR *)


    PROCEDURE PRINTERRORS ;
        VAR I : INTEGER ;

    BEGIN
        WRITE (' ':24)             ;
        FOR I := 1 TO BUFMAX DO
           BEGIN
              WRITE (ERRBUF [ I ]) ;
              ERRBUF [ I ] := ' '
           END ;
        WRITELN ;  WRITELN ;

        ERRPOS := 0 ;
        FOR I  := 1 TO ERRINX DO
           WITH ERRLIST [ I ] DO
              WRITELN (ERRWARN [ERRNUM < 500], ERRNUM : 3 ,
                   '...POSITION ',POS : 3, ' ***') ;

        WRITELN       ;
        ERRINX  := 0 ;
        LASTERR := 0
    END ; (* PRINTERRORS *)



 (********************************************************************
 *                                                                   *
 *)  PROCEDURE MARKSET (VAR CORRSET : SETS ; I : INTEGER) ;        (*
```

```
*               THIS PROCEDURE PERFORMS UNION OPERATION ON AN        *
*               ARRAY OF SETS                                        *
*                                                                    *
**********************************************************************)

        VAR J : INTEGER   ;
     BEGIN

        J              := I DIV SETSIZE   ;
        CORRSET [J] := CORRSET [J] + [ I MOD SETSIZE ]

     END ; (* MARKSET *)


(*********************************************************************
*                                                                    *
*)   FUNCTION  MEMBER (CORRSET : SETS ; I : INTEGER) :BOOLEAN ; (*
*               THIS PROCEDURE IS EQUIVALENT TO IN OPERATION         *
*               BUT FOR AN ARRAY OF SETS.                            *
*                                                                    *
**********************************************************************)

        VAR  J : INTEGER ;
     BEGIN

        J        := I DIV SETSIZE   ;
        MEMBER := (I MOD SETSIZE) IN CORRSET [ J ]

     END ; (* MEMBER *)


     FUNCTION NONTERMINAL (SPIND : SPRANGE) : BOOLEAN ;

     BEGIN
        WITH STABLE [ SPIND ], TTABLE [ TPTYPE ] DO
           NONTERMINAL := NOT TERMINAL
     END ;


     FUNCTION UNMARKED (VAR CORRSET : SETS; M : INTEGER) : BOOLEAN ;
        VAR J,K   : INTEGER ;

     BEGIN
        J     := M DIV SETSIZE ;
        K     := M MOD SETSIZE ;

        IF K IN CORRSET [ J ]
        THEN UNMARKED   := FALSE
        ELSE
           BEGIN
              UNMARKED := TRUE    ;
              CORRSET [J] := CORRSET [J] + [ K ]
           END
     END ; (* UNMARKED *)
```

```
      $PAGE

(*********************************************************************
*                                                                   *
*)  PROCEDURE EXPLAIN ;                                            (*
*             THIS PROCEDURE PRINTS DETAILED EXPLAINATION           *
*             OF ERROR CODES THAT HAS OCCURED.                      *
*                                                                   *
*********************************************************************)

        VAR   I,J : INTEGER ;

    BEGIN

        PAGE ;
        WRITELN (' ERROR EXPLAINATIONS ') ;

        FOR I := 1 TO 600 DO  BEGIN
           IF NOT  MEMBER (ERRDUPL,I) THEN
              J := 0 ELSE J := I ;
           CASE J OF
                0   : ;
                1   :  WRITELN ( '***ERROR ', J : 5,'....',
                 'ERROR IN SIMPLE TYPE  ' ) ;
                2   :  WRITELN ( '***ERROR ', J : 5,'....',
                 'IDENTIFIER EXPECTED             ' ) ;
                3   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''PROGRAM'' EXPECTED            ' ) ;
                4   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ''')'' EXPECTED                  ' ) ;
                5   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ''':'' EXPECTED                  ' ) ;
                6   :  WRITELN ( '***ERROR ', J : 5,'....',
                 'ILLEGAL SYMBOL                  ' ) ;
                7   :  WRITELN ( '***ERROR ', J : 5,'....',
                 'ERROR IN PARAMETER LIST  ' ) ;
                8   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''OF'' EXPECTED       ' ) ;
                9   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''('' EXPECTED                                           ' ) ;
               10   :  WRITELN ( '***ERROR ', J : 5,'....',
                 'ERROR IN TYPE                                            ' ) ;
               11   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''['' EXPECTED                                           ' ) ;
               12   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ''']'' EXPECTED                                           ' ) ;
               13   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''END'' EXPECTED                                         ' ) ;
               14   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ''';'' EXPECTED                                           ' ) ;
               15   :  WRITELN ( '***ERROR ', J : 5,'....',
                'INTEGER EXPECTED                                          ' ) ;
               16   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''='' EXPECTED                                           ' ) ;
               17   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''BEGIN'' EXPECTED                                       ' ) ;
               18   :  WRITELN ( '***ERROR ', J : 5,'....',                ' ) ;
                 'ERROR IN DECLARATION PART
               19   :  WRITELN ( '***ERROR ', J : 5,'....',
                 'ERROR IN FIELD LIST ') ;
               20   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ''','' EXPECTED  ' ) ;

               21   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ' MULOP ASSUMED HERE       ' ) ;

               22   :  WRITELN ( '***ERROR ', J : 5,'....',
                 ' ADDOP ASSUMED HERE  ' ) ;

               23   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''REPEAT'' EXPECTED ' ) ;

               24   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''LABEL'' ASSUMED HERE                                   ' ) ;

               25   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''CONST'' ASSUMED HERE                                   ' ) ;
               26   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''VAR '' ASSUMED HERE                                    ' ) ;

               27   :  WRITELN ( '***ERROR ', J : 5,'....',
                 '''TYPE '' ASSUMED HERE                                   ' ) ;
```

```
         28  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''.'' EXPECTED                                      ' ) ;

         50  :  WRITELN ( '***ERROR ', J : 5,'....',
          'ERROR IN CONSTANT                                   ' ) ;
         51  :  WRITELN ( '***ERROR ', J : 5,'....',
          ''':='' EXPECTED                                     ' ) ;
         52  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''THEN'' EXPECTED                                   ' ) ;
         53  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''UNTIL'' EXPECTED                                  ' ) ;
         54  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''DO''     EXPECTED                                 ' ) ;
         55  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''TO''/''DOWNTO'' TO EXPECTED                       ' ) ;
         56  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''IF'' EXPECTED                                     ' ) ;
         57  :  WRITELN ( '***ERROR ', J : 5,'....',
          '''FILE'' EXPECTED  ' ) ;
         58  :  WRITELN ( '***ERROR ', J : 5,'....',
          'ERROR IN FACTOR     ' ) ;
         59  :  WRITELN ( '***ERROR ', J : 5,'....',
          'ERROR IN VARIABLE  ' ) ;
        101  :  WRITELN ( '***ERROR ', J : 5,'....',
          'IDENTIFIER DECLARED TWICE  ' ) ;
        102  :  WRITELN ( '***ERROR ', J : 5,'....',
          'LOW BOUND EXCEEDS HIGH BOUND ' ) ;
        103  :  WRITELN ( '***ERROR ', J : 5,'....',
          'IDENTIFIER IS NOT OF APPROPRIATE CLASS  ') ;
        104  :  WRITELN ( '***ERROR ', J : 5,'....',
          'IDENTIFIER IS NOT DECLARED ') ;
        105  :  WRITELN ( '***ERROR ', J : 5,'....',
          'SIGN NOT ALLOWED HERE ') ;
        109  :  WRITELN ( '***ERROR ', J : 5,'....',
          'TYPE MUST NOT BE REAL ') ;
        113  :  WRITELN ( '***ERROR ', J : 5,'....',
          'INDEX TYPE MUST BE SCALAR OR SUBRANGE ') ;
        120  :  WRITELN ( '***ERROR ', J : 5,'....',
  'FUNCTION RESULT TYPE MUST BE SCALAR OR SUBRANGE ') ;
        115  :  WRITELN ( '***ERROR ', J : 5,'....',
          'BASE TYPE MUST BE SCALAR OR SUBRANGE_') ;
        148  :  WRITELN ( '***ERROR ', J : 5,'....',
          'SUBRANGE BOUNDS MUST BE SCALAR ' ) ;
        149  :  WRITELN ( '***ERROR ', J : 5,'....',
          'INDEX TYPE MUST NOT BE INTEGER ' ) ;
        169  :  WRITELN ( '***ERROR ', J : 5,'....',
          'ERROR IN BASE SET' ) ;
        173  :  WRITELN ( '***ERROR ', J : 5,'....',
          'TOO MUCH STORAGE REQUIRED FOR AN ARRAY ');
        201  :  WRITELN ( '***ERROR ', J : 5,'....',
          'ERROR IN REAL CONSTANT : DIGIT EXPECTED             ' ) ;
        160  :  WRITELN ( '***ERROR ', J : 5,'....',
          'PREVIOUS DECLARATION WAS NOT FORWARD  ') ;

        202  :  WRITELN ( '***ERROR ', J : 5,'....',
          'STRING CONSTANT CAN''T EXCEED SOURCE_LINE           ' ) ;
        203  :  WRITELN ( '***ERROR ', J : 5,'....',
          'INTEGER CONSTANT EXCEEDS RANGE' ) ;
        204  :  WRITELN ( '***ERROR ', J : 5,'....',
          'REAL CONSTANT EXCEEDS RANGE   ' ) ;
        206  :  WRITELN ( '***ERROR ', J : 5,'....',
          'INTEGER PART OF REAL CONSTANT EXCEEDS RANGE         ' ) ;
        250  :  WRITELN ( '***ERROR ', J : 5,'....',
          'TOO MANY NESTED SCOPE OF IDENTIFIERS ' ) ;
        251  :  WRITELN ( '***ERROR ', J : 5,'....',
          'TOO MANY NESTED PROCEDURES/FUNCTIONS ' ) ;

        255  :  WRITELN ( '***ERROR ', J : 5,'....',
          'LINE ERROR LIMIT EXCEEDED' ) ;

        205  :  WRITELN ( '***ERROR ', J : 5,'....',
          'NULL STRING NOT ALLOWED : '' '' ASSUMED' ) ;
        OTHERWISE
     END  (* CASE *)
   END
END ; (* EXPLAIN *)


PROCEDURE KEEP  ;
```

```
BEGIN

   MET := FALSE ; K := 1 ;
   WHILE (NOT MET) AND (K <= OKCOUNT) DO
      BEGIN
        MET :=(SPIND=LOSTLINKS [K].SPIND);
         K    := K + 1
      END  ;

   IF NOT MET THEN
      BEGIN
         OKCOUNT    := OKCOUNT + 1 ;
         IF    OKCOUNT    = 1 THEN
             LOSTLINKS [OKCOUNT].SPIND := SAVEDSP
         ELSE
             LOSTLINKS [OKCOUNT].SPIND := SPIND ;

         WITH LOSTLINKS [OKCOUNT] DO
            BEGIN
               NEW (TEMP ) ;
               TEMP @ := STACKTOP?
            END ;

         IF OKCOUNT > 1 THEN
            LOSTLINKS [OKCOUNT-1].TEMP
               @.  PREVIOUS :=
               LOSTLINKS [OKCOUNT].TEMP

      END
END  ;
```

```
                 $PAGE

(**********************************************************************
*                                                                     *
*)   PROCEDURE SCAN ;                                                (*
*              THIS ROUTINE IS THE LEXICAL ANALYZER. THE CLASS        *
*              AND VALUE (IF ANY) OF THE CURRENT SYMBOL IS            *
*              RETURNED IN THE RECORD VARIABLE 'TOKEN'.               *
*              SCAN     IS A HAND WRITTEN SCANNER EXCEPT FOR          *
*              IDENTIFIERS AND KEYWORDS FOR WHICH A SET OF            *
*              TRANSITION TABLES USED.                                *
*                                                                     *
**********************************************************************)


       LABEL 111             ;              (* RETURN ADDRESS           *)
       VAR   I,J,K           ;              (* COUNTERS                 *)
             EXPO,IVAL       : INTEGER;     (* FOR CONSTANT CONVERSION  *)
             P,PVAL,FAC      : REAL    ;    (* FOR CONSTANT CONVERSION  *)
             SIGN            ;              (* FOR NEGATIVE CONSTANTS   *)
             FINISHED        ;              (* FOR LOOP CONTROL         *)
             ERR             : BOOLEAN;     (* ERROR FLAG FOR CONSTANTS *)
             BAZ,STATE       : INTEGER;     (* FOR ID RECOGNITION       *)


       PROCEDURE PRINTLINE ;
          VAR I : INTEGER ;

       BEGIN    WRITE (' ') ;
          IF LNEST THEN
             BEGIN
                LNEST  := FALSE ;
                WRITE (NESTINCR  : 6)
             END
          ELSE  WRITE (' ' :6  )  ;
          IF RNEST THEN
             BEGIN
                RNEST  := FALSE ;
                WRITE  (NESTDECR  : 6)
             END
          ELSE  WRITE (' ' : 6 )  ;
          WRITE  ('  ') ;

          WRITE  ( ' ', CARDCNT : 5, '    ') ;
          FOR I  := 1 TO BUFMAX DO
             IF INBUF [ I ] IN ['A'..'Z']   THEN
                WRITE (LCASE [ INBUF [ I ]] )
             ELSE WRITE ( INBUF [ I ]  )    ;

          WRITELN
       END ; (* PRINTLINE *)


       PROCEDURE READLINE ;

          VAR     I : INTEGER ;

       BEGIN
          CHCNT    := 0
          IF  CARDCNT > 0 THEN PRINTLINE ;

          IF ERRINX > 0 THEN
             PRINTERRORS ;

          READLN (INPUT) ;
          IF EOF(INPUT)
          THEN
             BEGIN
                TOKEN.CLASS := EOFPGM       ;
                IF UNCLOSED <> NONE THEN
                   BEGIN
                      WRITELN (ABORTMES [UNCLOSED = COMMENT] ,
                              'STARTED AT LINE ',ABORTLINE : 3) ;
                      GOTO 1111
                   END ;
                IF CARDCNT = 0 THEN GOTO 1111 ELSE GOTO 111
             END
          ELSE
             BEGIN
                BUFLGTH := 0 ;
                WHILE (NOT EOLN(INPUT)) AND (BUFLGTH < BUFMAX) DO
          BEGIN
            BUFLGTH := BUFLGTH + 1 ;
            READ (INBUF [BUFLGTH])
          END ;

        I := BUFLGTH + 1 ;
        WHILE ( I <= BUFMAX ) DO
          BEGIN
            INBUF [I] := ' ' ;
            I := I + 1
          END ;
      END ;

    CARDCNT  := CARDCNT + 1
  END ; (* READLINE *)


  FUNCTION NEXTCH   : CHAR ;

  BEGIN
    IF CHCNT >= BUFLGTH
    THEN
      BEGIN
        NEXTCH := ' '  ;
        READLINE
      END
    ELSE
      BEGIN
        CHCNT    := CHCNT + 1 ;
        NEXTCH   := INBUF [CHCNT]
      END
  END ; (* NEXTCH  *)


  FUNCTION NEXTCHAR : CHAR ;

  BEGIN
    IF CHCNT >= BUFLGTH
    THEN
      NEXTCHAR   := ' '
    ELSE
      BEGIN
        CHCNT     := CHCNT + 1 ;
        NEXTCHAR  := INBUF [CHCNT]
      END
  END ; (* NEXTCHAR*)


BEGIN      (* SCAN *)
  REPEAT

    WHILE ( CH = ' ') DO
      BEGIN
        IF CHCNT >= BUFLGTH THEN
          READLINE ;

        CHCNT := CHCNT + 1 ;
        CH    := INBUF [CHCNT]
      END ;

    SYMSTART := CHCNT ;
    FINISHED := TRUE  ;

    CASE CH OF

      'A','B','C','D','E','F','G','H',
      'I','J','K','L','M','N','O','P',
      'Q','R','S','T','U','V','W','X',
      'Y','Z'      :

        BEGIN     (* ID RECOGNITION *)
          IDSTR         := DUMMYID  ;
          STATE         := ORD (CH) ;
          IDSTR [ 1 ]   := CH       ;

          CH  := NEXTCHAR     ;
          I   := 1            ;
          BAZ := BASE [STATE] + ORD (CH) ;
        WHILE STATE = CHECK [BAZ] DO
           BEGIN
              I          := I + 1                ;
              IDSTR [ I ] := CH                  ;
              CH         := NEXTCHAR             ;
              STATE      := NEXT [BAZ]           ;
              BAZ        := BASE [STATE] + ORD (CH)
           END ;

        IF CH IN IDCHARS THEN
           BEGIN
              STATE := 1              ;
              REPEAT
                 I   := I + 1         ;
                 IDSTR [I] := CH      ;
                 CH := NEXTCHAR
              UNTIL NOT (CH IN IDCHARS)
           END ;

        IDENTIFIER := IDSTR ;
        TOKEN := DEFAULT [STATE]  ;
        CASE TOKEN.CLASS OF
           RECORDSY , BEGINSY ,
           REPEATSY , CASESY  :
              IF (STABLE[STACKTOP].PTR].TPTYPE < 75)
                 AND (TOKEN.CLASS = CASESY) THEN ELSE
                 BEGIN
                    NEST  := NEST + 1 ;
                    IF NOT LNEST THEN BEGIN
                    NESTINCR:= NEST; LNEST:=TRUE END
                 END ;
           UNTILSY   , ENDSY    :
              BEGIN
                    RNEST := TRUE       ;
                    NESTDECR  := NEST   ;
                    NEST  := NEST - 1
              END ;
           OTHERWISE
        END   (*  CASE   *)
     END ;


'0','1','2','3','4','5','6','7',
'8','9' :

     BEGIN    (* DIGIT CONSTANTS *)
        TOKEN.CLASS := INTCONST    ;
        RVAL := 0 ;  ERR := FALSE ;

        REPEAT
           RVAL := RVAL * 10 + (ORD (CH) - ORD ('0')) ;
           CH   := NEXTCHAR
        UNTIL NOT (CH IN DIGITS)  ;

        IF CH = '.' THEN
           BEGIN
              CH := NEXTCHAR ;
              IF (CH = '.') OR (CH = ')')
              THEN
                 BEGIN
                    CHCNT := CHCNT - 1 ;
                    CH    := '.'
                 END
              ELSE
                 BEGIN
                    TOKEN.CLASS := REALCONST ;
                    IF NOT (CH IN DIGITS)
                    THEN ERROR (201)
                    ELSE
                       BEGIN
                          R        := 1          ;
                          REPEAT
                             R     := R * 10   ;
                             RVAL := RVAL * 10 +
                                (ORD (CH)-ORD('0')) ;
                             CH := NEXTCHAR
                          UNTIL NOT (CH IN DIGITS)    ;

                          RVAL := RVAL / R
                       END
                 END
           END ;
```

```
  IF (CH = 'E') THEN
    BEGIN
      TOKEN.CLASS := REALCONST ;
      SIGN        := FALSE     ;

      CH := NEXTCHAR           ;
      IF CH = '+'
      THEN CH := NEXTCHAR
      ELSE
        IF CH = '-' THEN
          BEGIN
            SIGN  := TRUE    ;
            CH    := NEXTCHAR
          END ;

      R := 0 ;
      IF NOT (CH IN DIGITS)
      THEN ERROR (201)
      ELSE
        REPEAT
          R := R * 10 + (ORD (CH)   ORD ('0')) ;
          CH := NEXTCHAR
        UNTIL NOT (CH IN DIGITS) ;

      IF R > MAXINT
      THEN ERROR (206)
      ELSE
        IF R <> 0 THEN
          BEGIN
            EXPO := TRUNC (R)      ;
            R    := 1 ; FAC := 10 ;

            REPEAT
              IF ODD (EXPO) THEN
                R := R * FAC ;
              IF FAC < SQRTREAL
              THEN
                BEGIN
                  FAC := SQR (FAC) ;
                  ERR := MAXREAL /
                         FAC <  R
                END
              ELSE ERR := TRUE       ;

              EXPO := EXPO DIV  2
            UNTIL (EXPO = 0) OR (ERR) ;

            IF EXPO <> 0
            THEN ERROR (207)
            ELSE
              IF SIGN
              THEN RVAL     := RVAL / R
              ELSE
                IF MAXREAL / R > RVAL
                THEN RVAL := RVAL * R
                ELSE ERROR (207)
          END
    END ;

  IF TOKEN.CLASS = INTCONST
  THEN
    BEGIN
      NEW (KONSPTR,INTGR) ;
      KONSPTR @. INTVAL := TRUE ;

      IF RVAL > MAXINT
      THEN BEGIN ERROR (203) ;
                 KONSPTR @. IVAL := MAXINT
           END
      ELSE KONSPTR @.IVAL := TRUNC (RVAL) ;
      TOKEN.CSTADR         := KONSPTR
    END
  ELSE
    BEGIN
      NEW (KONSPTR,REEL)       ;
      KONSPTR @. INTVAL :=FALSE;
      IF ERR THEN RVAL := 0    ;
      KONSPTR @. RVAL  := RVAL ;
      TOKEN.CSTADR     := KONSPTR
    END
END ;

[illegible] ;
```

```
    BEGIN
       LGTH        := 0          ;
       STRBUF      := DUMMYSTR   ;
       ABORTLINE   := CARDCNT    ;
       ERR         := FALSE      ;
       UNCLOSED    := STRINGS    ;

       REPEAT
          REPEAT
             CHCNT := CHCNT + 1      ;
             LGTH  := LGTH  + 1      ;
             CH    := INBUF [CHCNT]  ;
             IF NOT ERR THEN
                STRBUF [LGTH] := INBUF [CHCNT]
          UNTIL ( CH = '''') OR (CHCNT > BUFLGTH)
                OR (LGTH=STRGLGTH) ;

          IF CH = ''''
          THEN CH := NEXTCHAR
          ELSE
             BEGIN
                IF NOT ERR THEN
                   ERROR (202)  ;
                CH  := ''''     ;
                IF CHCNT > BUFLGTH THEN
                   READLINE     ;
                ERR := TRUE
             END
       UNTIL CH <> '''' ;


       UNCLOSED           := NONE      ;
       STRBUF [ LGTH ]    := ' '       ;
       LGTH               := LGTH - 1  ;

       IF  LGTH IN [ 0,1 ]
       THEN
          BEGIN
             IF LGTH = 0  THEN
                BEGIN  LSTH := 1 ;
                   STRBUF [1] := ' ' ;
                   ERROR (205)
                END ;

             TOKEN.CLASS   := CHARCONST ;
             NEW (KONSPTR) ;
             WITH KONSPTR@ DO        BEGIN

                CLASS        := INTGR ;
                IVAL         := ORD (STRBUF [1]) ;
                INTVAL       := TRUE      END ;
             TOKEN.CSTADR := KONSPTR
          END
       ELSE
          BEGIN
             TOKEN.CLASS := STRINGCONST ;
             NEW (KONSPTR,STRING) ;
             IF NOT ERR
             THEN
                WITH KONSPTR@ DO
                   BEGIN  INTVAL := FALSE ;
                      SLGTH := LGTH ;
                      SVAL  := STRBUF
                   END
             ELSE
                WITH KONSPTR@ DO
                   BEGIN  INTVAL := FALSE ;
                      SLGTH := 0 ;
                      SVAL  := DUMMYSTR
                   END ;
             TOKEN.CSTADR := KONSPTR
          END
    END ;

':' :

    BEGIN
       CH := NEXTCHAR ;
       IF CH = '='
       THEN
          BEGIN
             TOKEN.CLASS := ASGNOP ;
             CH            := NEXTCHAR
          END
       ELSE
          TOKEN.CLASS      := COLON
     END ;

'<' :

    BEGIN
       CH          := NEXTCHAR ;
       TOKEN.CLASS := RELOP ;
       IF CH = '='
       THEN
          BEGIN
             TOKEN.OP := LEOP ;
             CH       := NEXTCHAR
          END
       ELSE
          IF CH = '>'
          THEN
             BEGIN
                TOKEN.OP := NEOP ;
                CH       := NEXTCHAR
             END
          ELSE
             TOKEN.OP     := LTOP
    END ;

'>' :

    BEGIN
       CH          := NEXTCHAR ;
       TOKEN.CLASS := RELOP ;
       IF CH = '='
       THEN
          BEGIN
             TOKEN.OP := GEOP ;
             CH       := NEXTCHAR
          END
       ELSE  TOKEN.OP := GTOP
    END ;

'(' :

    BEGIN
       CH := NEXTCHAR ;
       IF CH = '.'
       THEN
          BEGIN
             TOKEN.CLASS := LBRACKET ;
             CH          := NEXTCHAR
          END
       ELSE
          IF CH = '*'
          THEN
             BEGIN
                UNCLOSED    := COMMENT    ;
                ABORTLINE   := CARDCNT    ;

                REPEAT
                   REPEAT
                   UNTIL NEXTCH = '*'
                UNTIL NEXTCH = ')' ;

                UNCLOSED    := NONE       ;
                CH          := NEXTCHAR   ;
                FINISHED    := FALSE
             END
          ELSE
             TOKEN.CLASS    := LPARENT
    END ;

'.' :

    BEGIN
       CH := NEXTCHAR ;
       IF CH = '.'
       THEN
          BEGIN
             TOKEN.CLASS := DOTDOT ;
             CH          := NEXTCHAR
          END
       ELSE
```

```
                  IF CH = ')'
                  THEN
                     BEGIN
                        TOKEN.CLASS  := LBRACKET ;
                        CH           := NEXTCHAR
                     END
                  ELSE
                     TOKEN.CLASS     := PERIOD
              END ;

          '$'  :   BEGIN I := 1 ; CH := NEXTCHAR ;
               IDSTR := DUMMYID ;
               WHILE CH IN ['A'..'Z'] DO
               BEGIN IDSTR[I] := CH ;I :=I+1; CH :=NEXTCHAR END;
               FINISHED := FALSE ; NAME := IDSTR ;
               PRNTABLE := NAME = 'PRINTABLES   '
              END  ;

          ',','[',']','@','¬',';','+',
          '-','*','/',')','='  :

              BEGIN
                 TOKEN  := CHRCLASS [CH] ;
                 CH     := NEXTCHAR
              END ;

          OTHERWISE

              BEGIN
                 ERROR (6) ;
                 CH        := NEXTCHAR ;
                 FINISHED  := FALSE
              END

        END  (* CASE *)

      UNTIL FINISHED ;
111:
    END ;      (* SCAN *)
```

```
                    $PAGE
(************************************************************
*                                                           *
*)   PROCEDURE RECOVAR ( SPINX : INTEGER ) ;               (*
*             THIS PROCEDURE TAKES CONTROL WHEN A SYNTAX ERROR  *
*             OCCURS TO RECOVAR IT. THE ALGORITHM DEPENDS ON    *
*             IRON'S TECHNIQUE FOR TOPDOWN PARSERS WITHOUT      *
*             BACKUP.                                           *
*                                                           *
*************************************************************)

       LABEL     22,56   ;
       VAR
          SUCCESSOR    : INTEGER ; (* SUCCESSIVE STABLE ENTRY    *)
          FOUND        ,           (* DEPENDS ON FUNCTION FIND   *)
          FND          ,           (* DEPENDS ON FUNCTION FIND   *)
          MET          ,           (* TEMPORARY                  *)
          REACHED      : BOOLEAN ; (* LOOP CONTROL FLAG          *)
          STEMP        ,           (* TEMPORARY PARSE STACK      *)
          CURRENT      ,           (*        .....               *)
          CURRPOS      : LINKS   ; (*      POINTERS              *)
          INSCONTRL    ,           (* CONTROLS INFINITE LOOPS    *)
          FINDCONTRL   : SETS    ; (* CONTROLS INFINITE LOOPS    *)


       FUNCTION FIND (SPINX : INTEGER) : BOOLEAN ;
          VAR FOUND : BOOLEAN ;
       BEGIN
          FOUND  := FALSE  ;
          WITH STABLE [ SPINX ], TTABLE [ TPTYPE ] DO
             CASE TERMINAL OF

                FALSE  :
                   BEGIN
                      IF UNMARKED (FINDCONTRL,TPTYPE) THEN
                         FOUND       := FIND (POINTER) ;

                      IF NOT FOUND THEN
                         IF (SUC  <> OK) AND (SUC   > SPINX) THEN
                            FOUND    := FIND (SUC )     ;

                      FIND           := FOUND
                   END  ;


                TRUE   :
                   BEGIN
                      IF CLASS = TOKEN.CLASS
                      THEN FIND     .    := TRUE
                      ELSE
                         BEGIN
                            IF (ALT <> FAIL) AND (ALT <> OK) THEN
                               FOUND    := FIND (ALT)   ;

                            IF NOT FOUND THEN
                               IF (SUC <> OK) AND (SUC> SPINX) THEN
                               FOUND    := FIND (SUC ) ;

                            FIND        := FOUND
                         END ;
                   END ;

             END   (* CASE *)
       END   ;     (* FIND *)


       PROCEDURE STRING ( SPINX : INTEGER ) ;

       BEGIN
          WITH STABLE [ SPINX ], TTABLE [ TPTYPE ] DO
             CASE TERMINAL OF

                FALSE  :
                   BEGIN
                      STRING (POINTER)   ;
                      IF SUC  <> OK THEN
                         STRING (SUC )
                   END ;
                TRUE   :
                   IF (ALT = FAIL) OR (SUC   = OK)
                   THEN
```

```
               BEGIN
                  RECIND                 := RECIND + 1  ;
                  RECOVARY [RECIND].CLASS
                                         := CLASS       ;

                  IF SUC  <> OK THEN
                     STRING (SUC )
               END
            ELSE
               IF ALT <> OK THEN
                  STRING (ALT)
      END   (* CASE      *)
END ;       (* STRING    *)


FUNCTION INSERT (SUCCESSOR : SPRANGE) : BOOLEAN ;

   LABEL  33     ;
   VAR    REACHED      : BOOLEAN ;
          SAVED        : INTEGER ;

BEGIN
   REACHED        := FALSE     ;
   FINDCONTRL     := EMPTYSET  ;

   WITH STABLE [SUCCESSOR], TTABLE [TPTYPE] DO
      CASE TERMINAL OF

         TRUE   :
            IF CLASS = TOKEN.CLASS
            THEN
               BEGIN
                  REACHED     := TRUE  ;

                  RECIND    := RECIND + 1 ;
                  RECOVARY [RECIND].CLASS := TOKEN.CLASS
               END
            ELSE
               IF (ALT = FAIL) OR (ALT = OK)
               THEN  GOTO 33
               ELSE
                  IF FIND (ALT)
                  THEN
                     BEGIN
                        SAVED       := RECIND   ;
                        REACHED     := INSERT (ALT) ;
                        IF NOT REACHED THEN
                           BEGIN
                              RECIND := SAVED + 1 ;
                              RECOVARY[RECIND].CLASS:=CLASS;
                              IF (SUC <> OK) AND
                                 (SUC > SUCCESSOR) THEN
                                REACHED := INSERT (SUC)
                           END
                     END
                  ELSE
               33 :
                        BEGIN
                           RECIND  := RECIND + 1 ;
                           RECOVARY [RECIND].CLASS := CLASS;
                        IF (SUC<>OK) AND (SUC>SUCCESSOR) THEN
                           REACHED := INSERT (SUC)
                        END   ;

        FALSE  :
            IF NOT FIND (POINTER)
            THEN
               BEGIN
                     STRING (POINTER) ;
                     IF SUC <> OK THEN
                        REACHED := INSERT (SUC)
               END
            ELSE
               IF UNMARKED (INSCONTRL,TPTYPE) THEN
                  BEGIN
                     FINDCONTRL := EMPTYSET ;
                     MARKSET (FINDCONTRL,TPTYPE) ;
                     IF (SUC <> OK) AND (SUC > SUCCESSOR)
                     THEN
                        IF FIND (SUC) THEN BEGIN
                           STRING (POINTER  ) ;
                           REACHED  := INSERT (SUC) END
                        ELSE REACHED := INSERT (POINTER)
                        ELSE  REACHED   := INSERT (POINTER) ;
                        IF NOT REACHED THEN
                           IF (SUC <> OK) AND (SUC > SUCCESSOR) TH
                              REACHED   := INSERT (SUC)
                     END
                 ELSE
                     BEGIN     SAVED   := RECIND ;
                        FINDCONTRL := EMPTYSET ;
                        IF  (SUC <> OK) AND (SUC>SUCCESSOR) THEN
                           IF FIND (SUC) THEN
                              BEGIN
                                 STRING (POINTER) ;
                                 REACHED := INSERT (SUC);
                                 IF NOT REACHED THEN
                                    BEGIN
                                       RECIND := SAVED ;
                                       REACHED:=INSERT(POINTER)
                                    END
                              END
                           ELSE
                        ELSE       STRING (POINTER)
                     END
      END ;  (* CASE       *)
   INSERT   := REACHED
END   ; (* INSERT *)
```

```
                 $PAGE
       BEGIN          (* RECOVAR *)

   (****************************************************************
   *                                                               *
   *   EACH MEMBER OF THE PARSE STACK SHOWS AN INCOMPLETE BRANCH.  *
   *   THE SYMBOL THAT CAUSED ERROR MUST BE IN THE INCOMPLETE PART *
   *   OF ONE OF THESE INCOMPLETE BRANCHES,OR IT MUST BE IN ONE OF *
   *   THE IMMEDIATE SUCCESSORS OF SPINX.                          *
   *   IF NOT, JUST SKIP THE CURRENT TOKEN  AND TRY NEXT UNTIL ONE *
   *   IS FOUND.                                                   *
   *                                                               *
   *******                                                    *****)

          MET := FALSE ;
          IF OKCOUNT > 0 THEN
             BEGIN
                FOR K := 1 TO OKCOUNT DO
                   WITH LOSTLINKS [K] .TEMP @ DO
                      IF PTR IN [16,17,18,19] THEN
                         BEGIN     MET := TRUE ;
                            FINDCONTRL := EMPTYSET ;
                   WITH STABLE [LOSTLINKS[K].SPIND],TTABLE[TPTYPE] DO
                      IF SUC <> OK THEN  IF FIND (SUC) THEN
                                  BEGIN
                                     STACKTOP@.PTR := 13 ;
                                     PREVIOUS := STACKTOP ;
                                     STACKTOP := LOSTLINKS[K].TEMP ;
                                     SPIND    := LOSTLINKS[K].SPIND ;
                                     SUCCESSOR := SPIND ;
                                     OKCOUNT := 0 ;
                                     ALTFLAG := FALSE ;
                                     GOTO 22
                                  END
                         END ;    IF MET THEN GOTO 56 ;
                LOSTLINKS [OKCOUNT]. TEMP @. PREVIOUS := STACKTOP ;
                WITH LOSTLINKS [ 1 ] DO
                   BEGIN
                      STACKTOP  := TEMP ;
                      SPINX     := SPIND
                   END ; ALTFLAG := FALSE ; OKCOUNT := 0 ;
                SPIND := SPINX
             END ;
56 :
          REPEAT
             FINDCONTRL   := EMPTYSET      ;
             WITH STABLE [ SPINX ] DO
                IF SUC  <> OK THEN  IF FIND (SUC) THEN
                   BEGIN
                      SUCCESSOR   := SPINX ;
                      SPIND       := SPINX ;

                      GOTO 22
                   END ;

             CURRPOS                := STACKTOP                        ;
             FOUND                  := FALSE                           ;
             END                    := FALSE                           ;
             CURRENT                := STACKTOP                        ;

             IF STACKTOP = NIL THEN
                BEGIN
                   WITH STABLE [SPINX], TTABLE [TPTYPE] DO
                      FOUND := CLASS = TOKEN.CLASS ;
                   SUCCESSOR := SPINX
                END ;

             WHILE (NOT FOUND) AND (CURRENT <> NIL) DO
                BEGIN
                   WITH STABLE [CURRENT @. PTR] DO
                      BEGIN
                         FINDCONTRL := EMPTYSET  ;
                         IF SUC  <> OK THEN  BEGIN
                            IF SUC  > CURRENT @. PTR THEN
                               MARKSET (FINDCONTRL, TPTYPE) ;
                            FOUND := FIND (SUC )  END
                      END  ;
                   IF FOUND THEN
                      BEGIN

                         STRING (SPINX) ;
```

```
                       SUCCESSOR := STABLE [CURRENT @. PTR].SUC
                    END ;
                 CURRPOS := CURRENT ; CURRENT := CURRENT @.PREVIOUS
              END ;   CURRENT := STACKTOP ;

           WHILE (NOT FOUND) AND (CURRENT <> NIL) DO
              BEGIN
                 WITH STABLE [CURRENT @. PTR],TTABLE[TPTYPE] DO
      IF ((POINTER =13) OR (POINTER =122)) AND (SPINX >= .50) THEN
                 ELSE     FOUND  := FIND (POINTER) ;

                 IF FOUND THEN
                    BEGIN
                       STACKTOP  := CURRENT ;
                    WITH STABLE[CURRENT@.PTR],TTABLE[TPTYPE] DO
                       BEGIN   SPIND := POINTER ;
                       SUCCESSOR := POINTER      END
                    END ;  CURRPOS   := CURRENT ;
                 CURRENT := CURRENT @. PREVIOUS
              END ;

           IF NOT FOUND
           THEN
              BEGIN
                 ERROR (6)    ;
                 SCAN         ;

                 IF TOKEN.CLASS = EOFPGM THEN
                    BEGIN
                       WRITELN ('**UNEXPECTED EOF INDICATOR   ',
                                '...COMPILATION ABORTED ***   ') ;
                       GOTO 1111
                    END
              END

        UNTIL FOUND  ;

 (******                                                          ******
 *                                                                     *
 *   THE CURRPOS NOW POINTS TO THE STABLE ENTRY WHICH ACCEPTS          *
 *   LAST SCANNED TOKEN, SAY T, AS A MEMBER OF ITS DEFINITION          *
 *   SO REMAINING STATEMENTS OF RECOVAR DETERMINES A TERMINAL          *
 *   STRING AND INSERTS IT JUST BEFORE T, SO THAT CONTINUATION         *
 *   OF THE PARSE CAN NOW BE CORRECTLY HANDLED                         *
 *                                                                     *
 *******                                                          *****)

        STEMP      := STACKTOP  ;


        WHILE (STEMP <> CURRPOS) DO
           WITH STEMP @ DO
              BEGIN
                 WITH STABLE [ PTR ] DO
                    IF SUC  <> OK THEN
                       STRING (SUC ) ;
                 STEMP := PREVIOUS
              END ;


22 :    INSCONTRL    := EMPTYSET ;
        WITH STABLE [SUCCESSOR],TTABLE [TPTYPE] DO
        IF TERMINAL THEN IF SUC <  SUCCESSOR  THEN
           BEGIN
              RECIND  := RECIND + 1 ;
              RECOVARY [RECIND].CLASS    := CLASS       ;
              SUCCESSOR := SUC
           END ;

        REACHED      := INSERT (SUCCESSOR) ;
        IF NOT REACHED THEN
           BEGIN
              WRITELN  ('*** COMPILER ERROR ***' ) ;
              STACKDUMP ;
              WRITELN ('*SPIND :',SPIND) ; HALT
           END
     END ;              (* RECOVAR *)
```

```
PROCEDURE PARSERROR ( SPINX : SPRANGE ) ;
    VAR USEDMSGS         : SETS           ;
        I, ERRNO         : INTEGER        ;
        CURRPOS          : LINKS          ;

BEGIN   USEDMSGS         := EMPTYSET      ;
    RECIND := 0 ;
    RECOVAR (SPINX)                               ;
    IF RECIND > 1 THEN
    FOR I := 1 TO RECIND - 1 DO
       BEGIN
          WITH RECOVARY [ I ] DO
          BEGIN
             ERRNO  := MISSING (CLASS)    ;

             IF (ERRNO>0) AND (UNMARKED(USEDMSGS,ERRNO)) THEN
                ERROR (ERRNO)
          END
       END ;


    CURRPOS                  := STACKTOP                  ;
    REPEAT
       WITH CURRPOS ^ DO
          BEGIN
             WITH STABLE [ PTR ] DO
                ERRNO := NONTRMSGS [TPTYPE]               ;
             IF ERRNO <> 0 THEN
                IF UNMARKED (USEDMSGS,ERRNO) THEN
                   ERROR (ERRNO)                          ;
             CURRPOS  := PREVIOUS
          END
    UNTIL CURRPOS = NIL
END ; (* PARSERROR *)
```

```
           $PAGE


    PROCEDURE PUSH (ELEMENT : SPRANGE ) ;
       VAR SAV  : LINKS ;
    BEGIN
       SAV              := STACKTOP  ;
       NEW  (STACKTOP)               ;
       WITH STACKTOP @ DO
          BEGIN
             PTR        := ELEMENT   ;
             PREVIOUS   := SAV
          END ;
       SAVE             := STACKTOP
    END ; (* PUSH *)

    PROCEDURE POP ;
       VAR I    : INTEGER ;


    BEGIN   IF ALTFLAG THEN KEEP ;
       SAVE         := STACKTOP                  ;
       STACKTOP     := STACKTOP @. PREVIOUS      ;

(*     MARK ( I )   ;
       RELEASE ( I )    *) DISPOSE (SAVE)

    END ; (* PUSH *)


    PROCEDURE SUCCESS ( MODE : INTEGER ) ;

    BEGIN
       WHILE (STABLE [SPIND].SUC  =OK) OR (MODE = 1) DO
          BEGIN
             MODE      := 0 ;
             IF STACKTOP = NIL THEN
             BEGIN
                   IF TOKEN.CLASS <> EOFPGM THEN
                      WRITELN ('**EOF EXPECTED***') ;
                   GOTO 1111
                END        ;
             SAVEDSP   := SPIND          ;
             SPIND     := STACKTOP @. PTR ;

(*           WRITELN ('***SEMANTICS(', SPIND : 5,')') ;    *)
             SEMANTICS(SPIND) ;

             POP
          END  ; (* WHILE *)
          SPIND        := STABLE [SPIND] . SUC
    END ; (* SUCCESS *)
```

```
             $PAGE


    BEGIN  (* MAIN PROGRAM *)
       PAGE    ;
       INITSCAN   ;
       INITPARSE ; STACKTOP := NIL ;
       ENTERSTDIDS ;
       SCAN       ; SPIND := 1 ;

(******                                                    ******
*                                                                *
*   THE PARSING ALGORITHM USED IN THIS IMPLEMENTATION IS A       *
*   TOPDOWN TECHNIQUE WITHOUT BACKUP. IT DEPENDS ON THE          *
*   ALGORITHM BY CHEATNEM AND SATTLEY                            *
*                                                                *
******                                                    ******)


       WHILE FOREVER DO
          BEGIN

             WHILE NONTERMINAL (SPIND) DO
                WITH STABLE [ SPIND ], TTABLE [ TPTYPE/] DO
                   BEGIN
                      PUSH (SPIND)   ;
                      SPIND := POINTER
                   END ;

             WITH STABLE [SPIND ], TTABLE [ TPTYPE ] DO
                IF CLASS = TOKEN.CLASS
                THEN
                   BEGIN
 (*         WRITELN ('***SEMANTICS(',SPIND:5,')') ;     *)
                      SEMANTICS(SPIND) ;
                      IF RECIND > 0
                      THEN
                         IF RIND = RECIND
                         THEN
                            BEGIN

                               RECIND              := 0      ;
                               RIND                := 0      ;
                               RECOVARED           := FALSE  ;

                               SCAN
                            END
                         ELSE
                            BEGIN
                               RIND                := RIND + 1 ;
                               IF RIND = RECIND THEN
                                  RECOVARED        := FALSE ;
                               TOKEN               := RECOVARY [RIND]
                            END
                      ELSE
                         SCAN ;
                      IF (TOKEN.CLASS = EOFPGM) AND
                         (STACKTOP <> NIL) THEN BEGIN
                         WRITELN ('**UNEXPECTED EOF INDICATOR..',
                              '  COMPILATION ABORTED ***') ;
                         GOTO 1111
                      END ;

                      ALTFLAG  := FALSE ;
                      OKCOUNT  := 0  ;
                      SUCCESS ( 0 )
                   END
                ELSE
                   IF ALT = FAIL
                   THEN
                      BEGIN
                         PARSERROR (SPIND) ;
                         RIND             := 1      ;
                         RECOVARED        := TRUE   ;
                         TOKEN            := RECOVARY [RIND]
                      END
                   ELSE
                      IF ALT = OK
                      THEN  BEGIN   ALTFLAG := TRUE ;
                               SUCCESS ( 1 )
                      END
```

```
                                        ELSE SPIND          := ALT
                   END ;  (* FOREVER LOOP *)

  1111 :     WRITELN ('          *** END OF ANALYSIS ***  ' ) ;
    WRITELN ('          ',CARDCNT : 6, ' RECORDS  ARE PROCESSED .') ;

    WRITELN ('          ', ERRWARNCNT [FALSE] : 6,' ERRORS DETECTED.');
                EXPLAIN
            END.  (* OF PASCAL ANALYZER *)

CONTAINS 4792 LINES AND 44 PROCEDURES
ARE ASCII FILES
5 .TIME USED: 237 SECONDS
=  NO ERRORS  NO WARNINGS  15.89 I#BANK  32768 D#BANK
ODE
L 06/27/83 21:01:12 (0)

S: 32537  TIME: 14.465  STORAGE: 5164/14/1/8236
3

/27/83 21:03:48


 001000 047441      19746 IBANK WORDS DECIMAL
 050000 150450      33065 DBANK WORDS DECIMAL
 006640



NT $MAIN$          001000 047441     050000 150450
          $(1)     001000 001174                                      07 AUG 78   09
          $(1)     001175 001570                                      07 AUG 78   09
                                                                      20 DEC 78   17
          $(1)     001571 001770                                      07 AUG 78   09
                                                                      17 JUL 78   13
          $(1)     001771 002241       $(0)      050000 050023        23 APR 82   12
          $(1)     002242 002261       $(0)      050024 050025        23 JUL 80   13
          $(3)     002262 002265
          $(1)     002266 002305       $(2)      050025 050026        18 MAR 80   14
          $(3)     002306 002312
          $(1)     002313 002363       $(2)      050027 050027        19 MAR 80   15
          $(3)     002364 002377
                                       $(2)      050030 050065        30 AUG 79   17
          $(1)     002400 002564       $(2)      050066 050077        23 OCT 78   11
          $(3)     002565 002601
          $(1)     002602 003357       $(2)      050100 050107        23 APR 82   10
          $(3)     003360 003415
          $(1)     003416 003431                                      23 OCT 78   11
          $(1)     003432 003605       $(2)      050110 050117        17 JUL 78   13
          $(1)     003606 003655                                      10 DEC 79   17
          $(3)     003656 003711
          $(1)     003712 004065                                      23 OCT 78   11
          $(3)     004066 004120
                                       $(0)      050120 050120        23 JUL 80   13
          $(1)     004121 004144                                      21 MAR 80   17
          $(1)     004145 005345       $(2)      050121 050234        11 DEC 79   16
          $(3)     005346 005372
          $(1)     005373 005502       $(2)      050235 050242        18 MAR 80   14
          $(3)     005503 005526
          $(1)     005527 006316       $(2)      050243 050343        23 APR 82   10
          $(3)     006317 006413
          $(1)     006414 006505       $(2)      050344 050344        16 JUL 80   11
          $(3)     006506 006543
          $(1)     006544 006613                                      20 MAR 80   18
          $(3)     006614 006631
          $(1)     006632 007770       $(2)      050345 050450        23 JUL 80   13
          $(3)     007771 010200
          $(1)     010201 011511                                      23 APR 82   10
          $(3)     011512 011714
          $(1)     011715 045600       $(036)    050451 150450        27 JUN 83   21
          $(3)     045601 047441
0  TIME: 23.177  STORAGE: 17792/4/040777/073777
*HCEM.PASCODE
AT11 06/27/83 21:04:03
```

APPENDIX B

TEST RUNS

```
          PROGRAM TEST1 (INPUT,OUTPUT) ;

      (*
              ...THIS DATA CONTAINS NO ERRORS
                 AND SYMBOL TABLE DUMP IS NOT REQUIRED...
      *)

          TYPE DAYS = (M,T,W,TH,FR,SA,S) ;
                WEEK = SET OF DAYS         ;

          VAR WK,WORK,FREE : WEEK ;
               D : DAYS ;  SS: WEEK ;

          PROCEDURE CHECK  ;
               VAR D : DAYS ;  RET :( K,L,M) ;
          BEGIN  WRITE (' ') ;
             FOR D := M TO SU DO
                IF D IN SS THEN WRITE ('X')
                ELSE WRITE ('0') ;
             WRITELN
          END ;

          BEGIN  WORK := [] ; FREE := [] ;
                WK := [M..SU] ;
                D := SA ; FREE := [D] + FREE +[SU] ;

                SS:= FREE ; CHECK  ;

                WORK := WK - FREE ;   SS:= CHECK; CHECK  ;

                IF FREE <= WK THEN WRITE ('D') ;
                IF WK >= WORK THEN WRITE ('K') ;
                IF NOT (WORK >= FREE) THEN WRITE (' JACK') ;
                IF [SA] <= WORK THEN WRITE (' FORGET IT') ;

                WRITELN
          END.
END OF ANALYSIS ***
42 RECORDS  ARE PROCESSED .
0 ERRORS DETECTED.
```

```
                        PROGRAM TEST2 (INPUT,OUTPUT) ;
                            $PRINTABLES
                      (*
                            ..THIS DATA IS SAME AS PROGRAM TEST1
                              BUT SYMBOL TAPLE DUMP IS NOW REQUIRED..

                      *)

                        TYPE DAYS = (M,T,W,TH,FR,SA,S) ;
                             WEEK = SET OF DAYS          ;

                        VAR WK,WORK,FREE : WEEK ;
                            D : DAYS ;  SS: WEEK ;

                        PROCEDURE CHECK  ;
                            VAR D : DAYS ;   RET :( K,L,M) ;
    ************************
    *    SYMBOL TABLE DUMP   *
    ************************

...INTERMEDIATE DUMP...


              ****IDENTIFIER****
   INTERNAL CODE=147724    NAME = D
  LLINK= *VIL*  RLINK=147674  NEXT=147674  IDPTR=150336
  CLASS=VARIABLE   ,ACTUAL      ,LEVEL=   1

               ****STRUCTURE****

  INTERNAL STRUCTURE CODE=150336      SIZE=1
  FORM=SCALAR     ,DECLARED   ,FIRST CONSTANT ID PTR=150120


               ****IDENTIFIER****
   INTERNAL CODE=147635    NAME = K
  LLINK= *NIL*  RLINK=147611  NEXT= *NIL*  IDPTR=147663
  CLASS=CONSTANT    ,CONST VALUE=0


               ****STRUCTURE****

  INTERNAL STRUCTURE CODE=147663      SIZE=1
  FORM=SCALAR     ,DECLARED   ,FIRST CONSTANT ID PTR=147565


               ****IDENTIFIER****
   INTERNAL CODE=147611    NAME = L
  LLINK= *NIL*  RLINK=147565  NEXT=147635  IDPTR=147663
  CLASS=CONSTANT    ,CONST VALUE=1


               ****IDENTIFIER****
   INTERNAL CODE=147565    NAME = M
  LLINK= *NIL*  RLINK= *NIL*  NEXT=147611  IDPTR=147663
  CLASS=CONSTANT    ,CONST VALUE=2


               ****IDENTIFIER****
   INTERNAL CODE=147674    NAME = RET
  LLINK=147635  RLINK= *NIL*  NEXT= *NIL*  IDPTR=147663
  CLASS=VARIABLE   ,ACTUAL      ,LEVEL=   1


   1                    BEGIN  WRITE (' ') ;
                           FOR D := M TO SU DO
                               IF D IN SS THEN WRITE ('X') ELSE WRITE(
                           WRITELN
          1             END ;

    ************************
    *    SYMBOL TABLE DUMP   *
    ************************

...INTERMEDIATE DUMP...


              ****IDENTIFIER****
   INTERNAL CODE=150633    NAME = BOOLEAN
  LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150641
  CLASS=TYPE
```

```
                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150641    SIZE=1
FORM=SCALAR     ,DECLARED  ,FIRST CONSTANT ID PTR=150672


                 ****IDENTIFIER****
  INTERNAL CODE=150716    NAME = CHAR
LLINK=150633  RLINK=150646  NEXT= *NIL*  IDPTR=150724
CLASS=TYPE

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150724    SIZE=1
FORM=SCALAR     ,STANDARD


                 ****IDENTIFIER****
  INTERNAL CODE=147755    NAME = CHECK
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR= *NIL*
CLASS=PROCEDURE ,ACTUAL     ,LEVEL=0,PARAMPTR=147750

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=147750    SIZE=0
FORM=PARAM LIST,FIRST PARAMETER PTR= *NIL*

                 ****IDENTIFIER****
  INTERNAL CODE=150024    NAME = D
LLINK=147755  RLINK= *NIL*  NEXT=150004  IDPTR=150336
CLASS=VARIABLE  ,ACTUAL      ,LEVEL=   0

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150336    SIZE=1
FORM=SCALAR     ,DECLARED  ,FIRST CONSTANT ID PTR=150120


                 ****IDENTIFIER****
  INTERNAL CODE=150347    NAME = DAYS
LLINK=150024  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150336
CLASS=TYPE

                 ****IDENTIFIER****
  INTERNAL CODE=150546    NAME = FALSE
LLINK=150347  RLINK=150170  NEXT= *NIL*  IDPTR=150641
CLASS=CONSTANT  BOOLEAN    ,CONST VALUE=FALSE


                 ****IDENTIFIER****
  INTERNAL CODE=150170    NAME = FR
LLINK= *NIL*  RLINK=150044  NEXT=150214  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=4


                 ****IDENTIFIER****
  INTERNAL CODE=150044    NAME = FREE
LLINK= *NIL*  RLINK= *NIL*  NEXT=150024  IDPTR=150076
CLASS=VARIABLE  ,ACTUAL      ,LEVEL=   0

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150076    SIZE=-29451204315
FORM=SET         ,ELSET PTR =150336

                 ****IDENTIFIER****
  INTERNAL CODE=150766    NAME = INTEGER
LLINK=150716  RLINK=150742  NEXT= *NIL*  IDPTR=150774
CLASS=TYPE

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150774    SIZE=1
FORM=SCALAR     ,STANDARD


                 ****IDENTIFIER****
  INTERNAL CODE=150310    NAME = M
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=0
```

```
                  ****IDENTIFIER****
   INTERNAL CODE=150742    NAME = MAXINT
LLINK=150310  RLINK=150730  NEXT= *NIL*  IDPTR=150774
CLASS=CONSTANT  INTEGER    ,CONST VALUE=34359738367


                  ****IDENTIFIER****
   INTERNAL CODE=150571    NAME = .NIL
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR= *NIL*
CLASS=CONSTANT


                  ****IDENTIFIER****
   INTERNAL CODE=150730    NAME = REAL
LLINK=150571  RLINK=150672  NEXT= *NIL*  IDPTR=150736
CLASS=TYPE

                  ****STRUCTURE****

INTERNAL STRUCTURE CODE=150736     SIZE=2
FORM=SCALAR     ,STANDARD


                  ****IDENTIFIER****
   INTERNAL CODE=150120    NAME = S
LLINK= *NIL*  RLINK= *NIL*  NEXT=150144  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=6


                  ****IDENTIFIER****
   INTERNAL CODE=150144    NAME = SA
LLINK=150120  RLINK=150004  NEXT=150170  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=5


                  ****IDENTIFIER****
   INTERNAL CODE=150004    NAME = SS
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150076
CLASS=VARIABLE   ,ACTUAL      ,LEVEL=   0

                  ****IDENTIFIER****
   INTERNAL CODE=150264    NAME = T
LLINK=150144  RLINK= *NIL*  NEXT=150310  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=1


                  ****IDENTIFIER****
   INTERNAL CODE=150621    NAME = TEXT
LLINK=150264  RLINK=150214  NEXT= *NIL*  IDPTR=150627
CLASS=TYPE

                  ****STRUCTURE****

INTERNAL STRUCTURE CODE=150627    SIZE=34359738367
FORM=FILE      ,FILE TYPE PTR=150724

                  ****IDENTIFIER****
   INTERNAL CODE=150214    NAME = TH
LLINK= *NIL*  RLINK= *NIL*  NEXT=150240  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=3


                  ****IDENTIFIER****
   INTERNAL CODE=150672    NAME = TRUE
LLINK=150621  RLINK=150240  NEXT= *NIL*  IDPTR=150641
CLASS=CONSTANT  BOOLEAN    ,CONST VALUE=TRUE


                  ****IDENTIFIER****
   INTERNAL CODE=150240    NAME = W
LLINK= *NIL*  RLINK=150110  NEXT=150264  IDPTR=150336
CLASS=CONSTANT    ,CONST VALUE=2


                  ****IDENTIFIER****
   INTERNAL CODE=150110    NAME = WEEK
LLINK= *NIL*  RLINK=150060  NEXT= *NIL*  IDPTR=150076
CLASS=TYPE

                  ****IDENTIFIER****
   INTERNAL CODE=150060    NAME = WK
LLINK= *NIL*  RLINK=150052  NEXT=150052  IDPTR=150076
CLASS=VARIABLE   ,ACTUAL      ,LEVEL=   0
```

```
                    ****IDENTIFIER****
   INTERNAL CODE=150052   NAME = WORK
 LLINK= *NIL*  RLINK= *NIL*  NEXT=150044   IDPTR=150076
 CLASS=VARIABLE   ,ACTUAL        ,LEVEL=    0


 1               22             BEGIN  WORK := [] ; FREE := [] ;
                 23                WK := [M..SU] ;
                 24                D := SA ; FREE := [D] + FREE +[SU] ;
                 25
                 26                SS:= FREE ; CHECK  ;
                 27
                 28                WORK := WK   FREE ;   SS:= CHECK; CHECK
                 29
                 30                IF FREE <= WK THEN WRITE ('O') ;
                 31                IF WK >= WORK THEN WRITE ('K') ;
                 32                IF NOT (WORK >= FREE) THEN WRITE (' JACK'
                 33                IF [SA] <= WORK THEN WRITE (' FORGET IT')
                 34
                 35                WRITELN
       1         36             END.
  *** END OF ANALYSIS ***
      36 RECORDS  ARE PROCESSED .
     0 ERRORS DETECTED.
```

```
                    1
                    2
                    3
                    4
                    5
                    6      PROGRAM ALLERRORS (INPUT OUTPUT) ;
                                                     1

*** ERROR   20...POSITION     1 ***

                    7      (*
                    8      THIS SAMPLE RUN CONTAINS ALOT OF ERRORS.
                    9      IT IS NOT A MEANINGFULL PROGRAM YET IT IS
                   10      DESIGNED TO TEST THE RECOVARY ALGORITHM
                   11      OF PUPASCAL.RECALL THAT NO SEMANTIC ANALYSIS
                   12      IS PERFORMED FOR STATEMENTS.
                   13      *)
                   14
                   15
                   16       CONST A = 9 ; B ='' ; C =365499999999993
                                              1       2

*** ERROR  205...POSITION     1 ***
*** ERROR  203...POSITION     2 ***

                   17       TYPE
      1            18        A1 = RECORD
                   19            N : REAL   ;
                   20            CASE M : INTEGER OF
                   21              1,2 : (CASE A : CHAR OF
                   22               'L' : (NN : CH) ;
                                                1

*** ERROR  104...POSITION     1 ***

                   23               'N','M' : (ZZ : 1..3  ) ;
                   24               'K','N' : (F : BOOLEAN)) ;
                                            1

*** ERROR  156...POSITION     1 ***

                   25              3,'A'; (FF : INTEGER  ;
                                        1

*** ERROR  111...POSITION     1 ***

                   26                CC : ARRAY [1..9] OF REAL),
                   27           END ;
                   28       VAR
                   29        AX  : A1  ;
                   30       I,J,,K : TEXT ;
                                1

*** ERROR    2...POSITION     1 ***
*** ERROR   18...POSITION     1 ***

                   31       I,AN   : BADTYPE   ;
                            1        2

*** ERROR  101...POSITION     1 ***
*** ERROR  104...POSITION     2 ***

                   32         SETS  : SET CHAR  ;
                                          1

*** ERROR    3...POSITION     1 ***
*** ERROR   10...POSITION     1 ***
*** ERROR   18...POSITION     1 ***

                   33         ARRAYS : ARRAY [INTEGER,BOOLEAN,REAL] OF
                                                     1            2

*** ERROR  149...POSITION     1 ***
*** ERROR  139...POSITION     2 ***

                   34             ARRAY [1..,3..9] OF CHAR ;
                                            1

*** ERROR   50...POSITION     1 ***
*** ERROR    1...POSITION     1 ***
*** ERROR   10...POSITION     1 ***
*** ERROR   18...POSITION     1 ***
```

```
                        35          ENUM :(E1,F2 E3) ;
                                                 1

*** ERROR     20...POSITION    1 ***
*** ERROR      1...POSITION    1 ***
*** ERROR     10...POSITION    1 ***
*** ERROR     18...POSITION    1 ***

                        36          POINT : B INTEGER ;
                        37
                        38
                        39         PROCEDURE P1 (VAR A,B : INTEGER ) ;
                        40          VAR K : FILEOF  CHAR ;
                                            1       2    3

*** ERROR    104...POSITION    1 ***
*** ERROR     23...POSITION    2 ***
*** ERROR      1...POSITION    2 ***
*** ERROR     10...POSITION    2 ***
*** ERROR     18...POSITION    2 ***
*** ERROR      6...POSITION    3 ***

       1                41         BEGIN
                        42          K = K + 9 ;
                                      1       2

*** ERROR      2...POSITION    1 ***
*** ERROR     11...POSITION    1 ***
*** ERROR     15...POSITION    1 ***
*** ERROR     12...POSITION    2 ***
*** ERROR     59...POSITION    2 ***

           1            43         END ;
                        44
                        45         FUNCTION P2 : A1 ;
                                                 1

*** ERROR    120...POSITION    1 ***

       1   1            46         BEGIN  END ;
                        47
       1                48       BEGIN (* MAIN BODY..
                        49                NO SEMANTICS CHECK IS DONE HERE *)
                        50        I := 9 ;
       2   2            51        WHILE (A  BEGIN J := J + 1; P1(A,B) END ;
                                             1

*** ERROR      4...POSITION    1 ***
*** ERROR     54...POSITION    1 ***
*** ERROR     58...POSITION    1 ***

                        52        JJ := P2
                        53        IF A > 3+9 THEN P1 (A,B) ;
                                  1

*** ERROR     14...POSITION    1 ***
*** ERROR     58...POSITION    1 ***

                        54        ELSE JJ := P2 + 1 ;
                                  1

*** ERROR     14...POSITION    1 ***
*** ERROR     55...POSITION    1 ***
*** ERROR     15...POSITION    1 ***
*** ERROR     52...POSITION    1 ***

                        55        H := ))) ;
                                       123 4

*** ERROR      2...POSITION    1 ***
*** ERROR     11...POSITION    1 ***
*** ERROR      9...POSITION    1 ***
*** ERROR     15...POSITION    1 ***
*** ERROR     58...POSITION    1 ***
*** ERROR     21...POSITION    2 ***
*** ERROR      2...POSITION    2 ***
*** ERROR     11...POSITION    2 ***
*** ERROR      9...POSITION    2 ***
*** ERROR    255...POSITION    4 ***

                        56
                        57        IF A THEN B ELSE IF MO THEN K := K +1
                        58        ELSE  IF B AND MO
```

```
                    59          WITH S, 3 DO K := K + 4
                                ↑
*** ERROR     52...POSITION    1 ***
*** ERROR     58...POSITION    1 ***

              ↑      60           END.
          *** END OF ANALYSIS ***
              60 RECORDS  ARE PROCESSED .
             69 ERRORS DETECTED.
```

```
 ERROR EXPLAINATIONS
***ERROR         1....ERROR IN SIMPLE TYPE
***ERROR         2....IDENTIFIER EXPECTED
***ERROR         4....')' EXPECTED
***ERROR         6....ILLEGAL SYMBOL
***ERROR         8....'OF' EXPECTED
***ERROR         9....'(' EXPECTED
***ERROR        10....ERROR IN TYPE
***ERROR        11....'[' EXPECTED
***ERROR        12....']' EXPECTED
***ERROR        14....';' EXPECTED
***ERROR        15....INTEGER EXPECTED
***ERROR        16....ERROR IN DECLARATION PART
***ERROR        20....',' EXPECTED
***ERROR        21.... MULOP ASSUMED HERE
***ERROR        28....'..' EXPECTED
***ERROR        50....ERROR IN CONSTANT
***ERROR        52....'THEN'  EXPECTED
***ERROR        54....'DO'    EXPECTED
***ERROR        56....'IF' EXPECTED
***ERROR        58....ERROR IN FACTOR
***ERROR        59....ERROR IN VARIABLE
***ERROR       101....IDENTIFIER DECLARED TWICE
***ERROR       104....IDENTIFIER IS NOT DECLARED
***ERROR       109....TYPE MUST NOT BE REAL
***ERROR       120....FUNCTION RESULT TYPE MUST BE SCALAR OR SUBRAN
***ERROR       149....INDEX TYPE MUST NOT BE INTEGER
***ERROR       203....INTEGER CONSTANT EXCEEDS RANGE
***ERROR       205....NULL STRING NOT ALLOWED : ' ' ASSUMED
***ERROR       255....LINE ERROR LIMIT EXCEEDED
@BRKPT PRINT$
```

```
                        PROGRAM RECORDS (INPUT,OUTPUT) ;

                     (* THIS TEST RUN IS USED TO SHOW CONTENTS OF SYMBOL
                        TABLE AFTER A VARIANT RECORD IS PROCESSED.
                     *)
                               $PRINTABLES
                        CONST   CHARMAX = 12 ;
                        PROCEDURE TESTREC ;
                           TYPE
                              ALFARANGE = 1 .. CHARMAX ;
                              AYRANGE = 1 .. 31 ;
                              ALFA      = PACKED ARRAY [ALFARANGE] OF CHAR ;
                              STATUS    = (MARRIED,WIDOWED,DIVORCED,SINGLE);
                              DATE      =
1                                RECORD    MO  : (JAN,FEB,MARCH,ETC) ;
                                          DAY : AYRANGE ;
                                         YEAR : INTEGER
      1                          END ;
1                             PERSON = RECORD
2     2                          NAME  : RECORD  FIRST,LAST : ALFA END ;
                                 SEX   : (MALE,FEMALE) ;
                                 BIRTH : DATE ;
                                 CASE MS : STATUS OF
                                    MARRIED,WIDOWED : (MDATE : DATE) ;
                                    DIVORCED        : (DDATE : DATE) ;
                                    SINGLE          : (INDEPT: BOOLEAN)
      1                       END  ;  (*  PERSON  *)
                        VAR PEOPLE  : PERSON ;
************************
   SYMBOL TABLE DUMP   *
************************

TERMEDIATE DUMP...


              ****IDENTIFIER****
  INTERNAL CODE=150171   NAME = ALFA
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTP=150153
CLASS=TYPE

              ****STRUCTURE****

INTERNAL STRUCTURE CODE=150153     SIZE=12
FORM=PARRAY : ,ELEMENT TYPE PTP=150724INDEX TYPE PTR=150227


              ****STRUCTURE****

INTERNAL STRUCTURE CODE=150227     SIZE=1
FORM=SUBRANGE   ,RANGE PTR=150774MIN=1MAX=12

              ****STRUCTURE****

INTERNAL STRUCTURE CODE=150774     SIZE=1
FORM=SCALAR      ,STANDARD


              ****IDENTIFIER****
  INTERNAL CODE=150247   NAME = ALFARANGE
LLINK=150171  RLINK=150221  NEXT= *NIL*  IDPTP=150227
CLASS=TYPE

              ****IDENTIFIER****
  INTERNAL CODE=150221   NAME = AYRANGE
LLINK= *NIL*  RLINK=150066  NEXT= *NIL*  IDPTP=150177
CLASS=TYPE

              ****STRUCTURE****

INTERNAL STRUCTURE CODE=150177     SIZE=1
FORM=SUBRANGE   ,RANGE PTR=150774MIN=1MAX=31

              ****IDENTIFIER****
  INTERNAL CODE=147762   NAME = DATE
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTP=147542
CLASS=TYPE
```

```
INTERNAL STRUCTURE CODE=147542      SIZE=-29451204315
FORM=RECORD       ,FIRST FIELD PTR=147750     VAR PTR= *NIL*

                  ****IDENTIFIER****
  INTERNAL CODE=147605     NAME = DAY
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150177
CLASS=FIELD

                  ****IDENTIFIER****
  INTERNAL CODE=147750     NAME = MO
LLINK=147605  RLINK=147565  NEXT= *NIL*  IDPTR=147737
CLASS=FIELD

                  ****STRUCTURE****

INTERNAL STRUCTURE CODE=147737      SIZE=1
FORM=SCALAR       ,DECLARED   ,FIRST CONSTANT ID PTR=147615

                  ****IDENTIFIER****
  INTERNAL CODE=147565     NAME = YEAR
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150774
CLASS=FIELD

                  ****IDENTIFIER****
  INTERNAL CODE=150016     NAME = DIVORCED
LLINK=147752  RLINK=147711  NEXT=150042  IDPTR=150114
CLASS=CONSTANT    ,CONST VALUE=2

                  ****STRUCTURE****

INTERNAL STRUCTURE CODE=150114      SIZE=1
FORM=SCALAR       ,DECLARED   ,FIRST CONSTANT ID PTR=147772

                  ****IDENTIFIER****
  INTERNAL CODE=147615     NAME = ETC
LLINK= *NIL*  RLINK= *NIL*  NEXT=147641  IDPTR=147737
CLASS=CONSTANT    ,CONST VALUE=3

                  ****IDENTIFIER****
  INTERNAL CODE=147665     NAME = FEB
LLINK=147615  RLINK=147366  NEXT=147711  IDPTR=147737
CLASS=CONSTANT    ,CONST VALUE=1

                  ****IDENTIFIER****
  INTERNAL CODE=147366     NAME = FEMALE
LLINK= *NIL*  RLINK= *NIL*  NEXT=147412  IDPTR=147440
CLASS=CONSTANT    ,CONST VALUE=1

                  ****STRUCTURE****

INTERNAL STRUCTURE CODE=147440      SIZE=1
FORM=SCALAR       ,DECLARED   ,FIRST CONSTANT ID PTR=147366

                  ****IDENTIFIER****
  INTERNAL CODE=147711     NAME = JAN
LLINK=147565  RLINK=147641  NEXT= *NIL*  IDPTR=147737
CLASS=CONSTANT    ,CONST VALUE=0

                  ****IDENTIFIER****
  INTERNAL CODE=147412     NAME = MALE
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=147440
CLASS=CONSTANT    ,CONST VALUE=0

                  ****IDENTIFIER****
  INTERNAL CODE=147641     NAME = MARCH
LLINK=147412  RLINK= *NIL*  NEXT=147665  IDPTR=147737
CLASS=CONSTANT    ,CONST VALUE=2
```

```
  INTERNAL CODE=150066  NAME = MARRIED
LLINK=150015  RLINK=150042  NEXT= *NIL*  IDPTR=150114
CLASS=CONSTANT   ,CONST VALUE=0


            ****IDENTIFIER****
  INTERNAL CODE=147064   NAME = PEOPLE
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=147076
CLASS=VARIABLE  ,ACTUAL      ,LEVEL=   1

            ****STRUCTURE****

INTERNAL STRUCTURE CODE=147076     SIZE=-29451204315
FORM=RECORD     ,FIRST FIELD PTR=147522   VAR PTR=147336

            ****IDENTIFIER****
  INTERNAL CODE=147356   NAME = BIRTH
LLINK= *NIL*  RLINK=147326  NEXT= *NIL*  IDPTR=147542
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147202   NAME = DDATE
LLINK= *NIL*  RLINK=147127  NEXT= *NIL*  IDPTR=147542
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147127   NAME = INDEPT
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150641
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147255   NAME = MDATE
LLINK=147202  RLINK= *NIL*  NEXT= *NIL*  IDPTR=147542
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147326   NAME = MS
LLINK=147255  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150114
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147522   NAME = NAME
LLINK=147355  RLINK=147451  NEXT= *NIL*  IDPTR=147457
CLASS=FIELD


            ****STRUCTURE****

INTERNAL STRUCTURE CODE=147457     SIZE=-29451204315
FORM=RECORD     ,FIRST FIELD PTR=147510   VAR PTR= *NIL*

            ****IDENTIFIER****
  INTERNAL CODE=147510   NAME = FIRST
LLINK= *NIL*  RLINK=147502  NEXT=147502  IDPTR=150153
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147502   NAME = LAST
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150153
CLASS=FIELD


            ****IDENTIFIER****
  INTERNAL CODE=147451   NAME = SEX
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=147440
CLASS=FIELD


            ****STRUCTURE****

INTERNAL STRUCTURE CODE=147336     SIZE=-29451204315
FORM=TAGFIELD  ,TAGFIELD PTR=147326,TAGTYPE  PTR=150114,FIRST VAR

            ****STRUCTURE****

INTERNAL STRUCTURE CODE=147152     SIZE=-29451204315
```

```
                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=147225      SIZE=-29451204315
FORM=VARIANT    ,NEXT VARIANT PTR=147300,SUBREC:    =147214,VARIANT

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=147300      SIZE=-29451204315
FORM=VARIANT    ,NEXT VARIANT PTR=147310,SUBREC:    =147267,VARIANT

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=147310      SIZE=-29451204315
FORM=VARIANT    ,NEXT VARIANT PTR= *NIL*,SUBREC:    =147267,VARIANT

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=147267      SIZE=-29451204315
FORM=RECORD     ,FIRST FIELD PTR=147255    VAR PTR= *NIL*

                    ****IDENTIFIER****
  INTERNAL CODE=147202    NAME = DDATE
LLINK= *NIL*  RLINK=147127   NEXT= *NIL*   IDPTR=147542
CLASS=FIELD


                    ****IDENTIFIER****
  INTERNAL CODE=147127    NAME = INDEPT
LLINK= *NIL*  RLINK= *NIL*   NEXT= *NIL*   IDPTR=150641
CLASS=FIELD


                    ****IDENTIFIER****
  INTERNAL CODE=147255    NAME = MDATE
LLINK=147202  RLINK= *NIL*   NEXT= *NIL*   IDPTR=147542
CLASS=FIELD


                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=147214      SIZE=-29451204315
FORM=RECORD     ,FIRST FIELD PTR=147202    VAR PTR= *NIL*

                    ****IDENTIFIER****
  INTERNAL CODE=147202    NAME = DDATE
LLINK= *NIL*  RLINK=147127   NEXT= *NIL*   IDPTR=147542
CLASS=FIELD


                    ****IDENTIFIER****
  INTERNAL CODE=147127    NAME = INDEPT
LLINK= *NIL*  RLINK= *NIL*   NEXT= *NIL*   IDPTR=150641
CLASS=FIELD


                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=147141      SIZE=-29451204315
FORM=RECORD     ,FIRST FIELD PTR=147127    VAR PTR= *NIL*

                    ****IDENTIFIER****
  INTERNAL CODE=147127    NAME = INDEPT
LLINK= *NIL*  RLINK= *NIL*   NEXT= *NIL*   IDPTR=150641
CLASS=FIELD


                    ****IDENTIFIER****
  INTERNAL CODE=147534    NAME = PERSON
LLINK=147064  RLINK= *NIL*   NEXT= *NIL*   IDPTR=147076
CLASS=TYPE

                    ****IDENTIFIER****
  INTERNAL CODE=147772    NAME = SINGLE
LLINK=147534  RLINK=150125   NEXT=150016   IDPTR=150114
CLASS=CONSTANT    ,CONST VALUE=3.


                    ****IDENTIFIER****
  INTERNAL CODE=150125    NAME = STATUS
LLINK= *NIL*  RLINK= *NIL*   NEXT= *NIL*   IDPTR=150114
CLASS=TYPE
```

```
                    ****IDENTIFIER****
  INTERNAL CODE=150042   NAME = WIDOWED
LLINK=147772  RLINK= *NIL*  NEXT=150066  IDPTR=150114
CLASS=CONSTANT    ,CONST VALUE=1


 1      1        34                BEGIN   END  ;
*************************
*    SYMBOL TABLE DUMP  *
*************************

...INTERMEDIATE DUMP...


                    ****IDENTIFIER****
  INTERNAL CODE=150633   NAME = BOOLEAN
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150641
CLASS=TYPE

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=150641    SIZE=1
FORM=SCALAR     ,DECLARED  ,FIRST CONSTANT ID PTR=150672


                    ****IDENTIFIER****
  INTERNAL CODE=150716   NAME = CHAR
LLINK=150633  RLINK=150646  NEXT= *NIL*  IDPTR=150724
CLASS=TYPE

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=150724    SIZE=1
FORM=SCALAR     ,STANDARD


                    ****IDENTIFIER****
  INTERNAL CODE=150335   NAME = CHARMAX
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150774
CLASS=CONSTANT  INTEGER    ,CONST VALUE=12

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=150774    SIZE=1
FORM=SCALAR     ,STANDARD


                    ****IDENTIFIER****
  INTERNAL CODE=150646   NAME = FALSE
LLINK=150335  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150641
CLASS=CONSTANT  BOOLEAN    ,CONST VALUE=FALSE


                    ****IDENTIFIER****
  INTERNAL CODE=150766   NAME = INTEGER
LLINK=150716  RLINK=150742  NEXT= *NIL*  IDPTR=150774
CLASS=TYPE

                    ****IDENTIFIER****
  INTERNAL CODE=150742   NAME = MAXINT
LLINK= *NIL*  RLINK=150730  NEXT= *NIL*  IDPTR=150774
CLASS=CONSTANT  INTEGER    ,CONST VALUE=34359738367


                    ****IDENTIFIER****
  INTERNAL CODE=150571   NAME = NIL
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR= *NIL*
CLASS=CONSTANT


                    ****IDENTIFIER****
  INTERNAL CODE=150730   NAME = REAL
LLINK=150571  RLINK=150672  NEXT= *NIL*  IDPTR=150736
CLASS=TYPE

                    ****STRUCTURE****

INTERNAL STRUCTURE CODE=150736    SIZE=2
```

```
                 ****IDENTIFIER****
  INTERNAL CODE=150304   NAME = TESTREC
LLINK= *NIL*  RLINK= *NIL*  NEXT= *NIL*  IDPTR= *NIL*
CLASS=PROCEDURE ,ACTUAL      ,LEVEL=0,PARAMPTR=150277

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150277     SIZE=0
FORM=PARAM LIST,FIRST PARAMETER PTR= *NIL*

                 ****IDENTIFIER****
  INTERNAL CODE=150621   NAME = TEXT
LLINK=150304  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150627
CLASS=TYPE

                 ****STRUCTURE****

INTERNAL STRUCTURE CODE=150627     SIZE=34359738367
FORM=FILE        ,FILE TYPE PTR=150724

                 ****IDENTIFIER****
  INTERNAL CODE=150672   NAME = TRUE
LLINK=150521  RLINK= *NIL*  NEXT= *NIL*  IDPTR=150641
CLASS=CONSTANT  BOOLEAN.  ,CONST VALUE=TRUE


1              35                BEGIN
       1       35                END.
   *** END OF ANALYSIS ***
         36 RECORDS  ARE PROCESSED .
        0 ERRORS DETECTED.
```

APPENDIX C
SYNTAX OF STANDARD PASCAL

1. <program> ::= <program heading><body>.

2. <program heading>:
::= PROGRAM IDENT (<ident list>);

3. <ident list> ::= IDENT {, IDENT}$_o$

4. <body>::= <declarations><routine declarations>
<compound statement>

5. <declarations> ::= <label declarations>
<constant definitions>
<type definitions>
<var declarations>

6. <label declarations>
::= LABEL INTCONST {, INTCONST}$_o$; | empty

7. <constant definitions>
::= CONST {IDENT = <constant>;}$_1$ | empty

8. <constant>::= INTCONST | REALCONST | IDENT | ADDOP INTCONST | ADDOP
REALCONST | ADDOP IDENT|STRINGCONST | CHARCONST

9. <type definitions>::= TYPE {IDENT = <type>;}$_1$ | <empty>

10. <type> ::= IDENT | PACKED <structured type>
SET OF <simple type> |
ARRAY [<index list>] OF <type>|
RECORD <field list> END |
FILE OF <type> |
<simple type>

11. <structured type>::= SET OF <simple type> |
ARRAY [<index list>] OF <type> |
RECORD <field list> END |
FILE OF <type>

12. <index list> ::= <simple type> {,<simple type>}$_0$

13. <simple type>::= (<ident list>) | IDENT <subrange tail> |
<constant> .. <constant>

14. <subrange tail>::= .. <constant> | <empty>

15. <field list>::= <fixed part><variant part>

16. <fixed part>::= IDENT {, IDENT}$_0$: <type> {; <ident list>: <type>}$_0$
| <empty>

17. <variant part> ::= CASE IDENT <tagtype> OF <variant list>

18. <tagtype> ::= : IDENT | <empty>

19. <variant list>::= <variant> {; <variant>}$_0$

20. <variant>::= <constant list>: (<field part>

21. <field part>::=)| <field list>)

22. <constant list>::= <constant> {,<constant>}$_0$

23. <var declarations>::= VAR {IDENT {, IDENT}$_0$: <type>}$_1$ | <empty>

24. <routine declarations> ::=
PROCEDURE IDENT <formal parameter section>; <routine tail>;|
FUNCTION IDENT <formal parameter section>: IDENT; <routine tail>; | <empty>

25. <routine tail>::= FORWARD | <body>

26. <formal parameter section>::= (<formal parameters>) | <empty>

27. <formal parameters>::= <formal parameter> {; <formal parameter>}$_0$

28. <formal parameter>::=
PROCEDURE <identlist> | FUNCTION <parameter group>
VAR <parameter group> | <parameter group>

29. <parameter group>::= <identlist>: IDENT

30. <compound statement>::= BEGIN <statement list> END

31. <statement list>::= <statement>{; <statement>}$_o$

32. <statement>::= INTCONST: <unlabelled statement> |
<unlabelled statement>

33. <unlabelled statement>::=
BEGIN <statement list> END |
IF <expression> THEN <statement> <else part> |
CASE <expression> OF <case list> END |
WHILE <expression> DO <statement> |
REPEAT <statement list> UNTIL <expression> |
FOR IDENT := <expression><for tail> |
WITH <record-var list> DO <statements> |
<simple statement>

34. <else part>::= ELSE <statement> | <empty>

35. <for tail>::= TO <expression> DO <statement> |
DOWNTO <expression> DO <statement>

36. <simple statement> ::= GOTO INTCONST |
IDENT <assignment> | <empty>

37. <assignment>::= (<expression list>) |
<var tail> := <expression>

38. <var tail>::= @ <dot part> |
[<expression list>] <dot part> |
<dot part>

39. <dot part>::= .<variable> | <empty>

40. <variable>::= IDENT <var tail>

41. <record var list>::= <variable>{,<variable>}$_{o}$

42. <expression list>::= <expression>{, <expression>}$_{o}$

43. <expression>::= <simple expression>{RELOP <simple expression>}$_{o}$

44. <simple expression>::= ADDOP <term> {ADDOP <term>}$_{o}$ |
<term> {ADDOP <term>}$_{o}$

45. <term>::= <factor>{MULOP <factor>}$_{o}$

46. <factor>::= INTCONST | REALCONST | STRINGCONST |
CHARCONST | [<subset list> |
NOT <factor> | (<expression>) |
IDENT <factor tail>

47. <factor tail>::= (<expression list>) | <var tail>

48. <subset list>::=] | <element list>]

49. <element list>::= <element> {, <element>}$_{o}$

50. <element>::= <expression>< range part>

51. <range part>::= .. <expression> | <empty>

52. <case list>::= <case list element> {; <case list element>}

53. <case list element>::= <constant list>: <statement>

54. <empty>:: =

BIBLIOGRAPHY

1- AHO,A., ULLMAN,J., "Principles of Compiler Design", Addison-Wesley Publishing Company, 1977.

2- GRIES,D., "Compiler Construction For Digital Computers", John Wiley and Sons, Inc., 1971.

3- HOROWITZ,E., SAHNI,S., "Fundamentals of Data Structures", Pitman Publishing Limited, 1976.

REFERENCES NOT CITED

1- Ammann, "The Method of Structured Programming Applied to the Development of a Compiler", in International Computing Symposium. 1974, Amsterdam: North Holland Publishing Co., pp.93-99.

2- Vensen,K., Wirth,N., "PASCAL-User Manual and Report", 2nd ed., New York: Springer-Verlag, 1978.

3- Wirth,N., "The Design of a PASCAL Compiler", Software Practice and Experience, 1, No.4, 309-334, 1971.