2. Let w contain delimiters.

$$\begin{aligned}\{\ &\langle ";" \quad , \text{ SEMICOLON}\rangle,\\ &\langle "'" \quad , \text{ QUOTE}\rangle,\\ &\langle ":" \quad , \text{ COLON}\rangle,\\ &\langle "(" \quad , \text{ LPAREN}\rangle,\\ &\langle ")" \quad , \text{ RPAREN}\rangle,\\ &\langle "\{" \quad , \text{ COMBGN}\rangle,\\ &\langle "\}" \quad , \text{ COMEND}\rangle,\\ &\langle "*" \quad , \text{ ORDINARY}\rangle,\\ &\langle "." \quad , \text{ ORDINARY}\rangle,\\ &\langle \backslash e \quad , \text{ ENDFILE}\rangle,\\ &\langle "(*" \quad , \text{ COMBGN}\rangle,\\ &\langle "*)" \quad , \text{ COMEND}\rangle,\\ &\langle " ::= " \ , \text{ ASSIGN}\rangle\ \}\end{aligned}$$

Note the obvious fact that any string consisting of exactly one non-white character is a token.

The essence of lexical analysis is captured in LEX which produces the token sequence of z in the context of a token sequence T already produced. Recall that 00 denotes the empty token sequence, and $\circ$ denotes concatenation of token sequences.

Definition of LEX

1. LEX(T, " ") ::= 00.
2. LEX(T, "%" | y) ::= LEX(T, y).
3. Let z be free of leading white space. Then LEX(T, z) ::= $t \circ$ LEX(T $\circ\ t, x$), where $z = w \mid x$, and w is the longest prefix of z such that TKN(w) is defined. The token t is TKN(w) unless (i) TKN(w) $\neq$ COMEND and T = S $\circ$ COMBGN $\circ$ ORDINARY*, or (ii) TKN(w) $\neq$ QUOTE and T = S $\circ$ QUOTE $\circ$ ORDINARY*, for some S free of unmatched QUOTES. In the latter two cases, t = ORDINARY.

Definition of TKNSEQ

TKNSEQ(z) ::= LEX(00, z).

Since syntactically correct Pascal programs have one of the delimiters immediately following reserved words, we do not risk non-recognition of such words by ignoring other conventional delimiters (such as operators). Lexical analysis performed by a typical Pascal compiler otherwise matches with LEX except when dealing with comments and strings. In compilers, comments are simply swallowed and the strings are returned as tokens. For our purpose here,, however, the layout of comments is important. It would seem logical then to split a comment into three tokens, namely, COMBGN, the comment contained, followed by COMEND. Since comments can span several lines, this decision would complicate the definitions of functions given in subsequent sections. Thus, we define LEX(T, z) based on the longest prefix of z that is a word, and change the token to ORDINARY if T has an unmatched COMBGN or QUOTE.

For example, the string "(*(*)" is broken into words as "(*" | "(*" | ")" giving the token sequence T1 $\circ$ T3 is a reduced token sequence of T1 $\circ$ T2 T3 if T2 is the token "'*doesn*''*t it*'" is tokenized as "'" | "*doesn*" | "'" | "'" | "*t*" | "*it*" | "'". Note, however, that our line splitting rules do not split a Pascal string that was contained in one source