The role of the Lexical analyzer

Vasudev Ravula

The lexical analyzer is the first phase of a compiler. Its task is to read the input characters and produce as output a ssequence of tokens that the parser uses for syntax analysis. This interaction, is commonly implemented by making the lexical analyzer be a routine or a coroutine of the parser. Upon receiving a "generate next token" command from parser, the lexical analyzer reads inpput characters until it can identify the next token.

Since the lexical analyzer is the part of the compiler that reads the source text, it may also perform certain secondary tasks at the user interface. One such task is stripping out from the source program comments and white space in the form of blank, tab and new line characters. Another is correlating error messages from the compiler with the source program. For example, the lexical analyzer may keep track of the number of newline characters seen, so that a line number can be associated with an error message. In some compilers, the lexical analyzer is in charge of making a copy of the source program with the error messages marked in it. If the source language supports some macro preprocessor functions, then these preprocessor functions may also be implemented as lexical analysis takes place.

Sometimes, lexical amalyzers are divided into a cascade of two phases, the first called "scanning", and the second "lexical analysis." The scanner is responsible for doing simple tasks, while the lexical analyzer phase does the more complex operations. For example, a Pascal compiler might use a scanner to eliminate blanks from the input.

0.1 Issues in Lexical Analysis

There are several reasons for seperating the analysis of compiling into lexical analysis and parsing.

1. Simple design is perhaps the most inportant consideration. The separation of lexical analysis form syntax analysis often allows us to simplify one or the other of these phases. For example, a parser embodying the convertions for comments and white space is significantly more complex athan one that can assume comments and white spave have already been removed by the lexical analyzer. If we are designing a new language, separating the lexical and syntactic cinventions can lead to a cleaner overall language design.

2. Compiler efficency is improved. A separate lexical analyzer allows us to construct a specialized and potentially more efficient processor for the task. A large amount of time is spent on reading the sourcce program and partitioning it into tokens. Specialized buffering techniques for reading input characters and processing tokens can significantly speed up the performance of a compiler.

3. Compier portablility is enhanced. Input alphaber peculiarities and device-specific anomalies can be restricted to the lexical analyzer. The representation of special or non-standard symbols, such as ”’ in pascal, can be isolated in lexical analyzer.

Specialized tools have been designed to help automate the construction of lexical analyzer and the parsers when they are separated.