Thursday, February 7, 2019

Lexical Analysis

The Role of The Lexical Analyzer

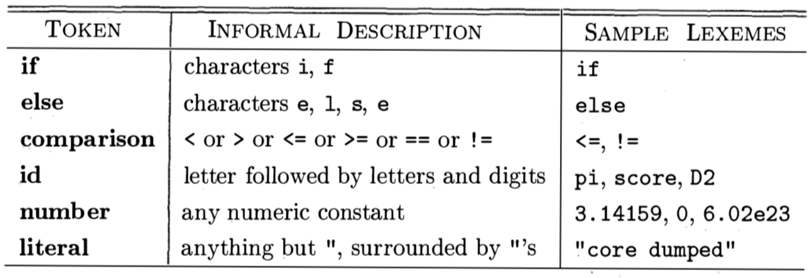
- the main task of the lexical analyzer is to read the input characters of the source program,program, group them into lexemes and produce as output a sequence of tokens for each lexeme in the source program.
- The stream of tokens is sent to the parser for syntax analysis
- When the lexical analyzer discovers a lexeme , it needs to enter that lexeme into the symbol table
- The separation of lexical and syntatic(parser)analyisis allows us to simplify at least one of these tasks.It is also more efficient and the portability is enhanced.

Terms

-A token is a pair consisting of a token name and an optional attribute value.(keywords)

-A pattern is a description of the form that the lexemes of a token may take.

-A lexeme is a sequence of charm caters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token

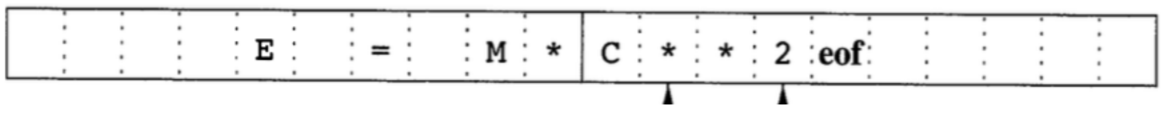


Token	Informal Description	Sample Lexemes
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Thursday, February 7, 2019

Input Buffering

-Because of the amount of time taken to process characters and the large number of characters that must be processed during the completion of a large source program, specialized buffering techniques have been developed like an scheme with two buffers, one buffer at the begin of the lexeme and the other buffer forward to check further tokens for patterns.



-

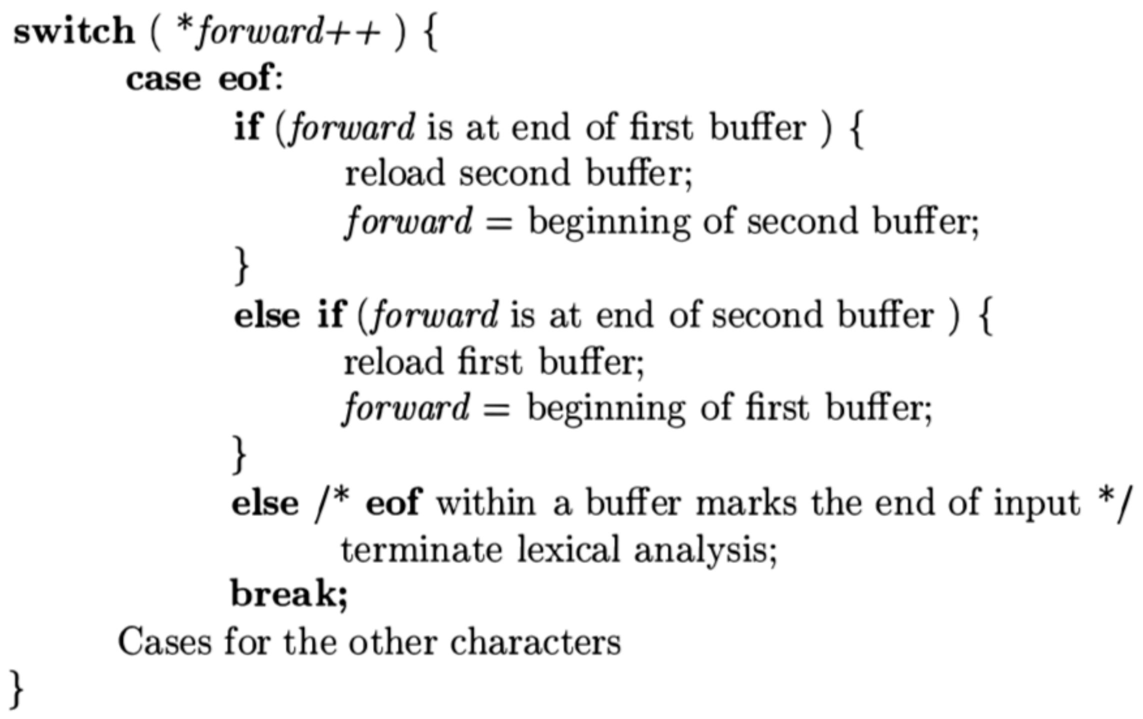
1. Pointer `lexemeBegin`, marks the beginning of the current lexeme, whose extent we are attempting to determine.
2. Pointer `forward` scans ahead until a pattern match is found; the exact strategy whereby this determination is made will be covered in the balance of this chapter.

Once the next lexeme is determined, `forward` is set to the character at its right end. Then, after the lexeme is recorded as an attribute value of a token returned to the parser, `lexemeBegin` is set to the character immediately after the lexeme just found. In Fig. 3.3, we see `forward` has passed the end of the next lexeme, `**` (the Fortran exponentiation operator), and must be retracted one position to its left.

Advancing `forward` requires that we first test whether we have reached the end of one of the buffers, and if so, we must reload the other buffer from the input, and move `forward` to the beginning of the newly loaded buffer. As long as we never need to look so far ahead of the actual lexeme that the sum of the lexeme's length plus the distance we look ahead is greater than N, we shall never overwrite the lexeme in its buffer before determining it.

Thursday, February 7, 2019

Code Example of implementation



```
switch ( *forward++ ) {
        case eof:
                if (forward is at end of first buffer ) {
                        reload second buffer;
                        forward = beginning of second buffer;
                }
                else if (forward is at end of second buffer ) {
                        reload first buffer;
                        forward = beginning of first buffer;
                }
                else /* eof within a buffer marks the end of input */
                        terminate lexical analysis;
                break;
        Cases for the other characters
}
```