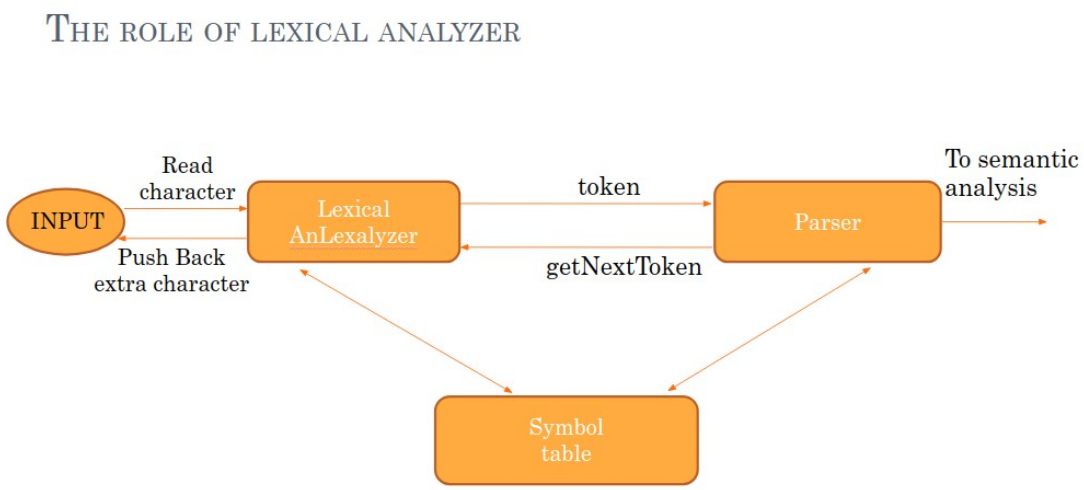
LEXICAL ANALYSIS

Lexical analysis is the first phase of a compiler.

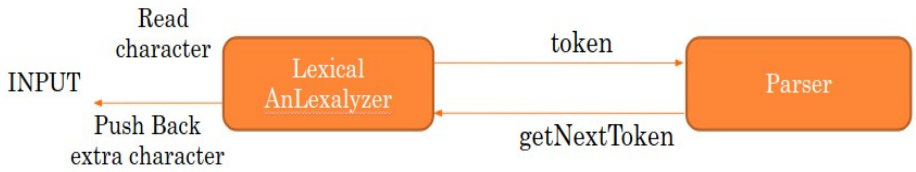
It takes the modified source code from language preprocessors that are written in the form of sentences.

The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.



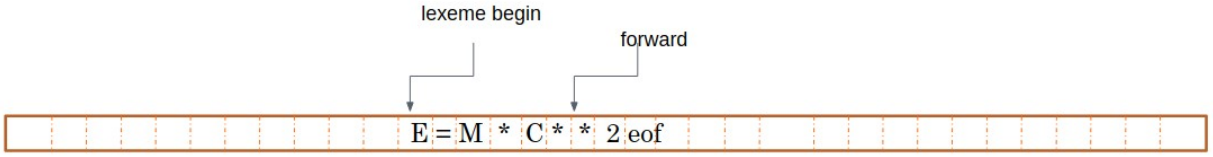
INPUT BUFFERING

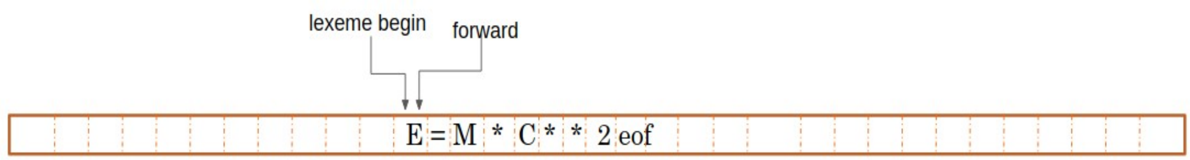
Sometimes lexical analyzer needs to look ahead some symbols to decide about the token to return.
These extra characters has to be pushed back to the input becouse it can be begining of the next lexeme.
The implementation of reading and pushing back char is usually done by setting up an input buffer.

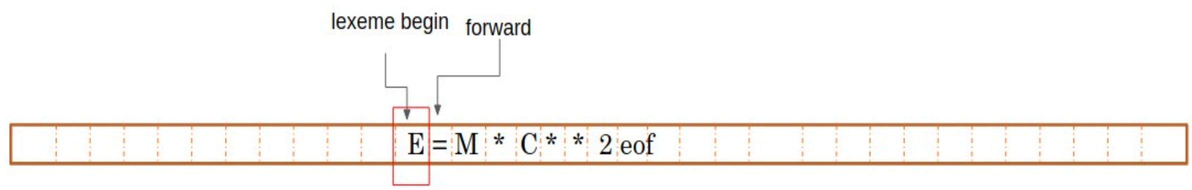


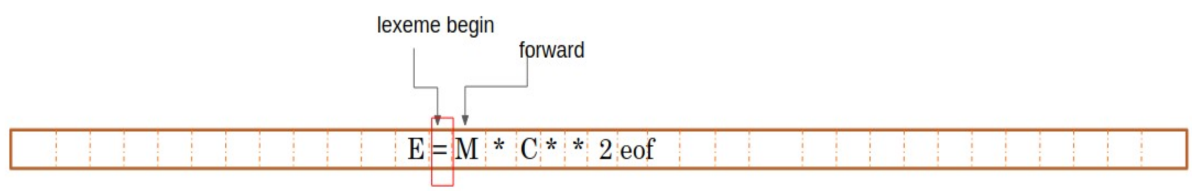
Sometimes lexical analyzer needs to look ahead some symbols to decide about the token to return
These extra characters has to be pushed back to the input becouse it can be beginning of the next lexeme.
The implementation of reading and pushing back char is usually done by setting up an input buffer.
For input buffering two pointers are maintained:

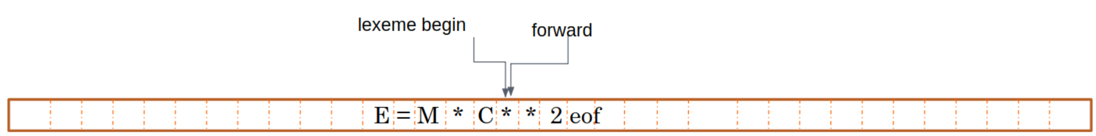
- lexeme begin
- forword

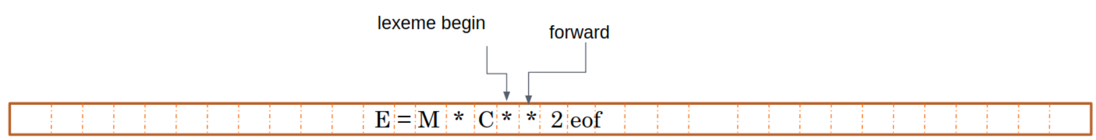


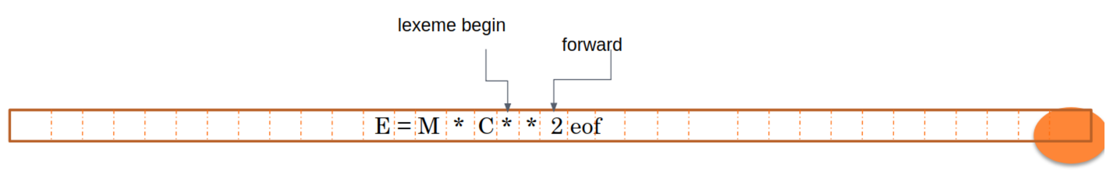

lexeme begin
forward
E = M * C * * 2 eof


lexeme begin
forward
E = M * C * * 2 eof


lexeme begin
forward
E = M * C * * 2 eof


lexeme begin
forward
E = M * C * * 2 eof


lexeme begin
forward
E = M * C * * 2 eof


lexeme begin
forward
E = M * C * * 2 eof

2 BUFFER TECHNIQUE

Because of the amount of time required to process char and the large no. of characters processed during the compilation of large source program specialized buffer techniques are used to reduce the overhead required to process a single input character.
we use a buffer divided into 2 N-character halves.
each buffer is of the same size N, (N is usually the size of a disk block.
The first buffer and second buffer are scanned alternately. when end of current buffer is reached the other buffer is filled. the only problem with this method is that if length of the lexeme is longer than length of the buffer then scanning input cannot be scanned completely.
Initially both the lexeme begin(lb) and forward pointer(fp) are pointing to the first character of first buffer. Then the fp moves towards right in search of end of lexeme. as soon as blank character is recognized, the string between lb and fp is identified as corresponding token. to identify, the boundary of first buffer end of buffer character should be placed at the end first buffer.
Similarly end of second buffer is also recognized by the end of file(eof) mark present at the end of second buffer. when fp encounters first eof, then one can recognize end of first buffer and hence filling up second buffer is started. in the same way when second eof is obtained then it indicates of second buffer. alternatively both the buffers can be filled up until end of the input program and stream of tokens is identified. This eof character introduced at the end is calling Sentinel which is used to identify the end of buffer.