Input Buffering – Compiler Design

BY DINESH THAKUR

• To ensure that a right lexeme is found, one or more characters have to be looked up beyond the next lexeme.

• Hence a two-buffer scheme is introduced to handle large lookaheads safely.

• Techniques for speeding up the process of lexical analyzer such as the use of sentinels to mark the buffer end have been adopted.

There are three general approaches for the implementation of a lexical analyzer:

(i) By using a lexical-analyzer generator, such as lex compiler to produce the lexical analyzer from a regular expression based specification. In this, the generator provides routines for reading and buffering the input.

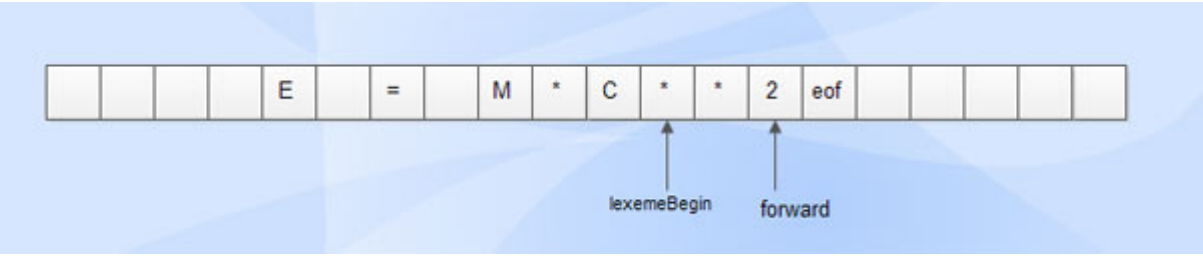
(ii) By writing the lexical analyzer in a conventional systems-programming language, using I/O facilities of that language to read the input.

(iii) By writing the lexical analyzer in assembly language and explicitly managing the reading of input.

Buffer Pairs

Because of large amount of time consumption in moving characters, specialized buffering techniques have been developed to reduce the amount of overhead required to process an input character.

Fig shows the buffer pairs which are used to hold the input data.



Scheme

• Consists of two buffers, each consists of N-character size which are reloaded alternatively.

• N-Number of characters on one disk block, e.g., 4096.

• N characters are read from the input file to the buffer using one system read command.

• *eof* is inserted at the end if the number of characters is less than N.

Pointers

Two pointers *lexemeBegin* and *forward* are maintained.

lexeme Begin points to the beginning of the current lexeme which is yet to be found.

forward scans ahead until a match for a pattern is found.

• Once a lexeme is found, *lexemebegin* is set to the character immediately after the lexeme which is just found and *forward* is set to the character at its right end.

• Current lexeme is the set of characters between two pointers.

Disadvantages of this scheme

• This scheme works well most of the time, but the amount of lookahead is limited.

• This limited lookahead may make it impossible to recognize tokens in situations where the distance that the forward pointer must travel is more than the length of the buffer.

(eg.) DECLARE (ARGl, ARG2, . . . , ARGn) in PL/1 program;

• It cannot determine whether the DECLARE is a keyword or an array name until the character that follows the right parenthesis.

Sentinels

• In the previous scheme, each time when the forward pointer is moved, a check is done to ensure that one half of the buffer has not moved off. If it is done, then the other half must be reloaded.

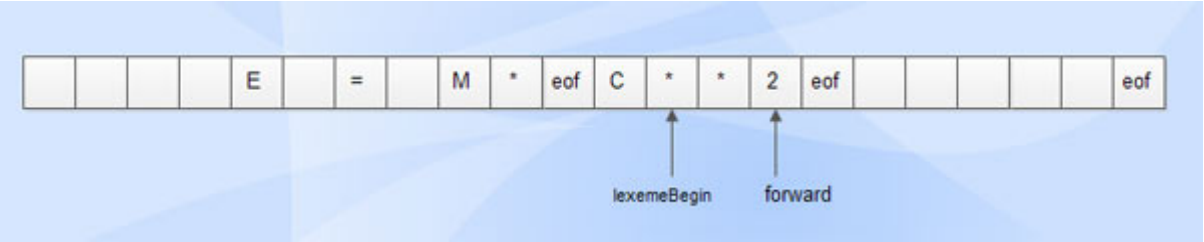
• Therefore the ends of the buffer halves require two tests for each advance of the forward pointer.

Test 1: For end of buffer.

Test 2: To determine what character is read.

• The usage of sentinel reduces the two tests to one by extending each buffer half to hold a sentinel character at the end.

• The sentinel is a special character that cannot be part of the source program. *(eof* character is used as sentinel).



Advantages

• Most of the time, It performs only one test to see whether forward pointer points to an *eof*.

• Only when it reaches the end of the buffer half or *eof*, it performs more tests.

• Since N input characters are encountered between *eofs,* the average number of tests per input character is very close to 1.