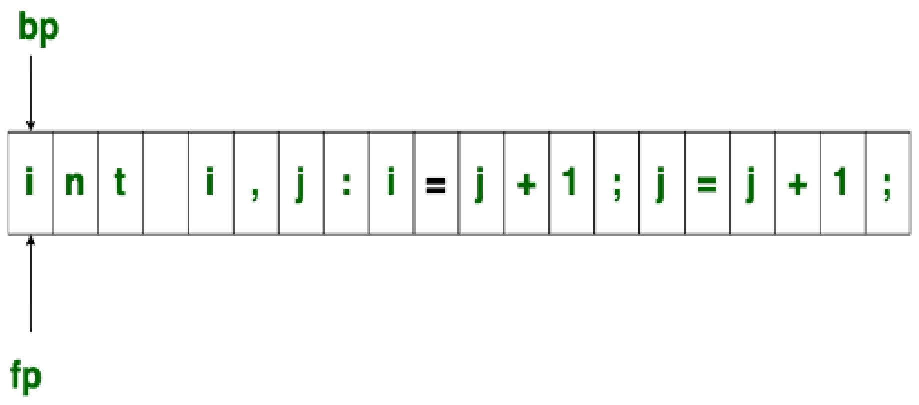
Input Buffering in Compiler Design

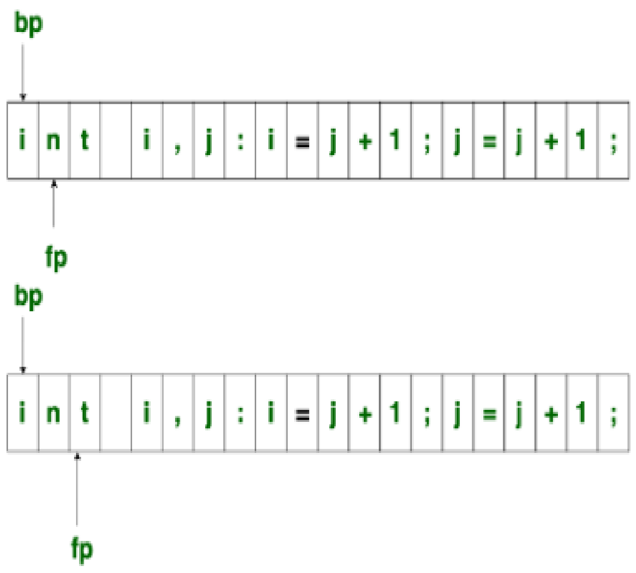
Last Updated: 21-11-2019

The lexical analyzer scans the input from left to right one character at a time. It uses two pointers begin ptr(**bp**) and forward to keep track of the pointer of the input scanned.



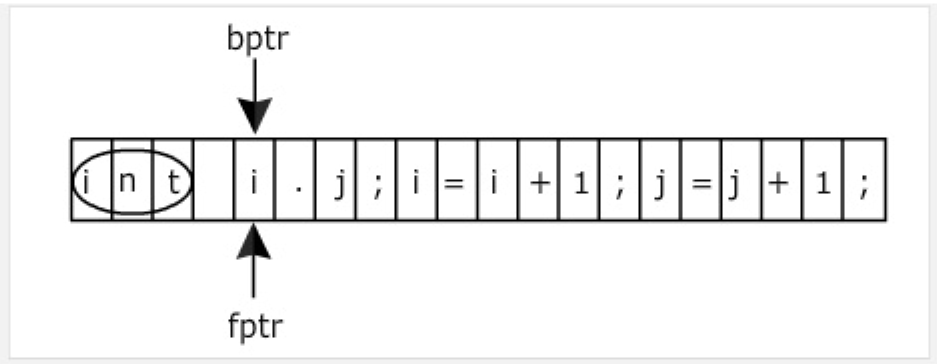
Initial Configuration

Initially both the pointers point to the first character of the input string as shown below



Input Buffering

The forward ptr moves ahead to search for end of lexeme. As soon as the blank space is encountered, it indicates end of lexeme. In above example as soon as ptr (fp) encounters a blank space the lexeme “int” is identified.



Input Buffering

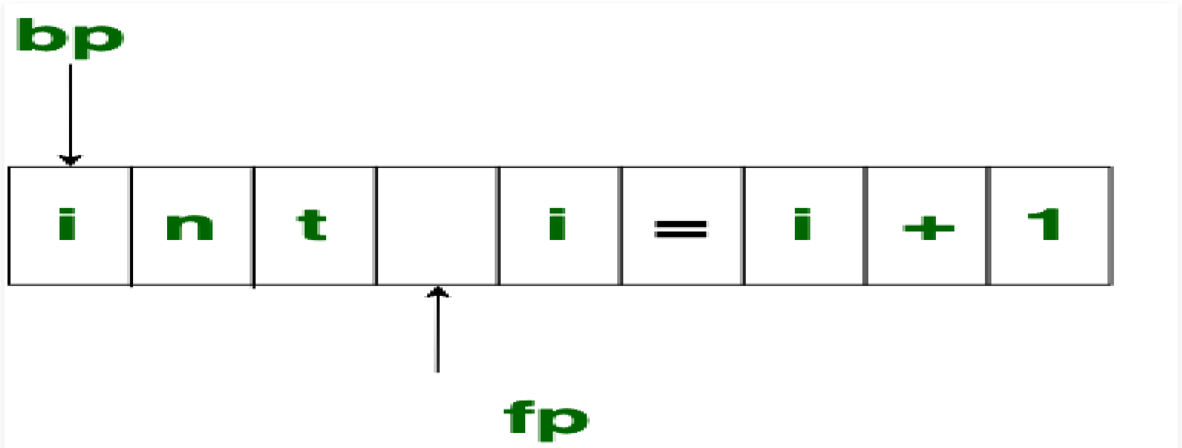
The input character is read from secondary storage. But reading in this way from secondary storage is costly. Hence buffering technique is used A block of data is first read into a buffer, and then scanned by lexical analyzer

There are two methods used in this context

1. One Buffer Scheme
2. Two Buffer Scheme

One Buffer Scheme:

In this scheme, only one buffer is used to store the input string. But the problem with this scheme is that if lexeme is very long then it crosses the buffer boundary, to scan rest of the lexeme the buffer has to be refilled, that makes overwriting the first part of lexeme.



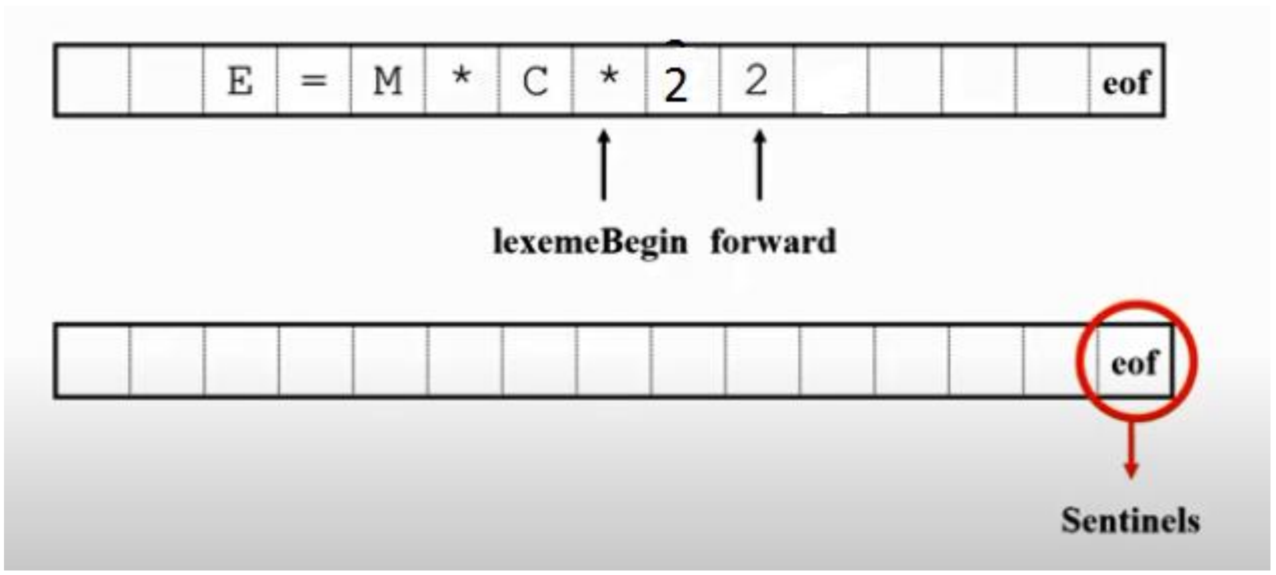
One buffer scheme storing input string

Two Buffer Scheme:

To overcome the problem of one buffer scheme, in this method two buffers are used to store the input string. The first buffer and second buffer are scanned alternately. When end of current buffer is reached the other buffer is filled.

Initially both the bp and 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 bp 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 buffer 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.


E
=
M
*
C
*
2
2
eof
lexemeBegin
forward
eof
Sentinels