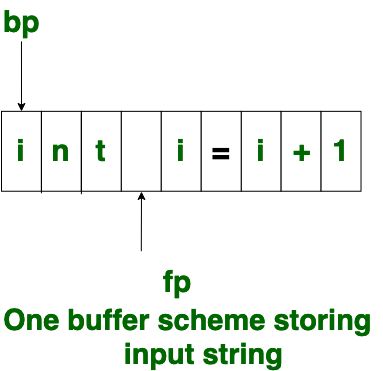
1. **One Buffer Scheme:**

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 of lexeme.



1. **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. 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 ‘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.

