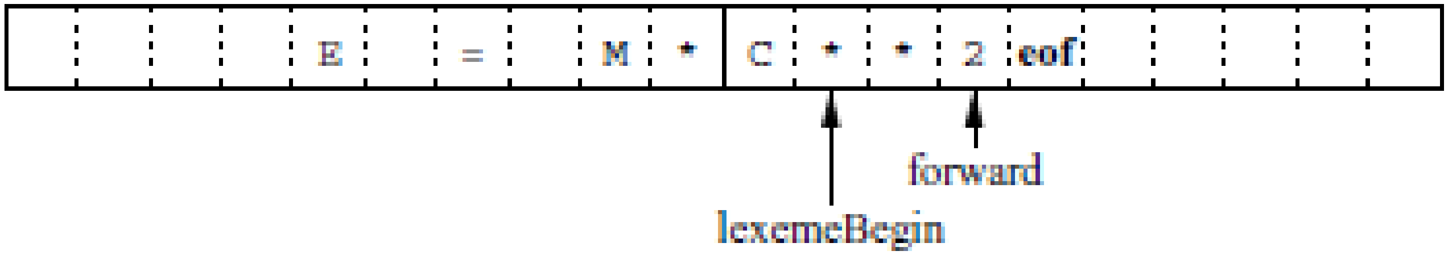
Input Buffering

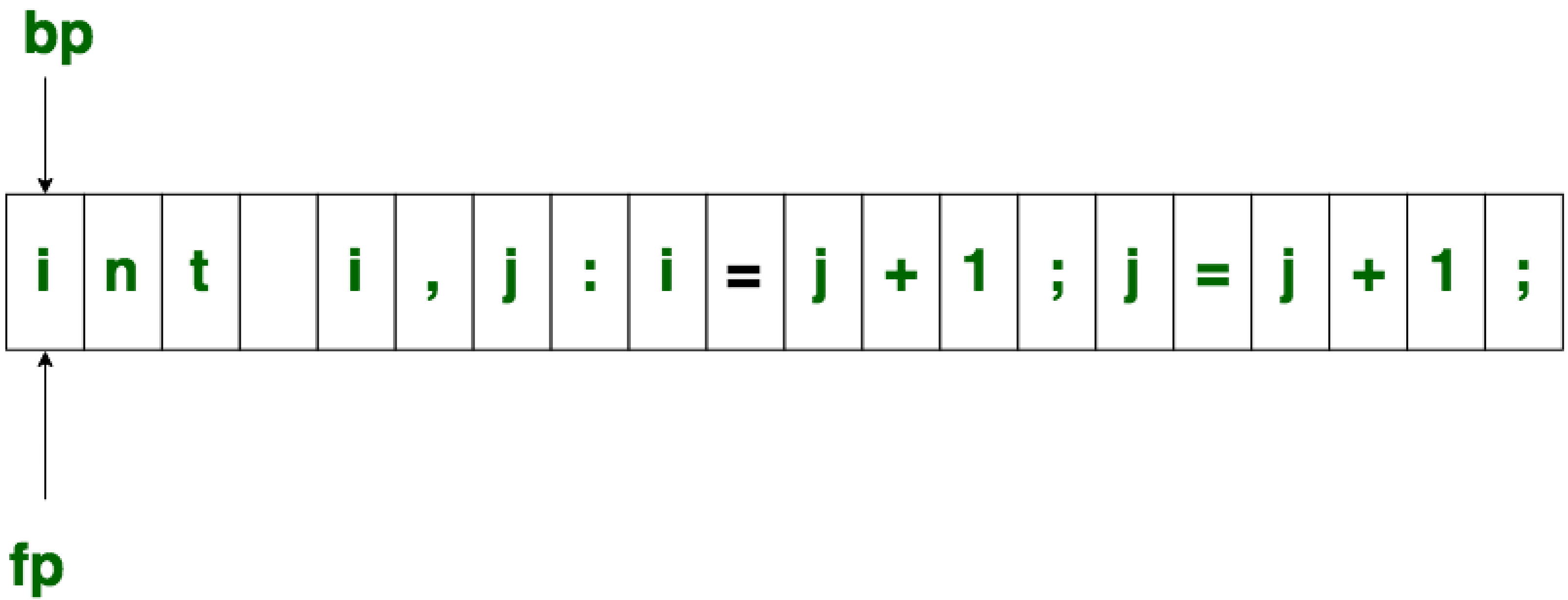
- Input Buffering
  - Lexical Analysis – Right lexeme → one /more characters look up
  - Left to Right → backward pointer and forward pointer
  - Disk read operation – costly → Buffer



  - Pointers to buffer pair
    - Lexeme begin
    - forward

Input Buffering

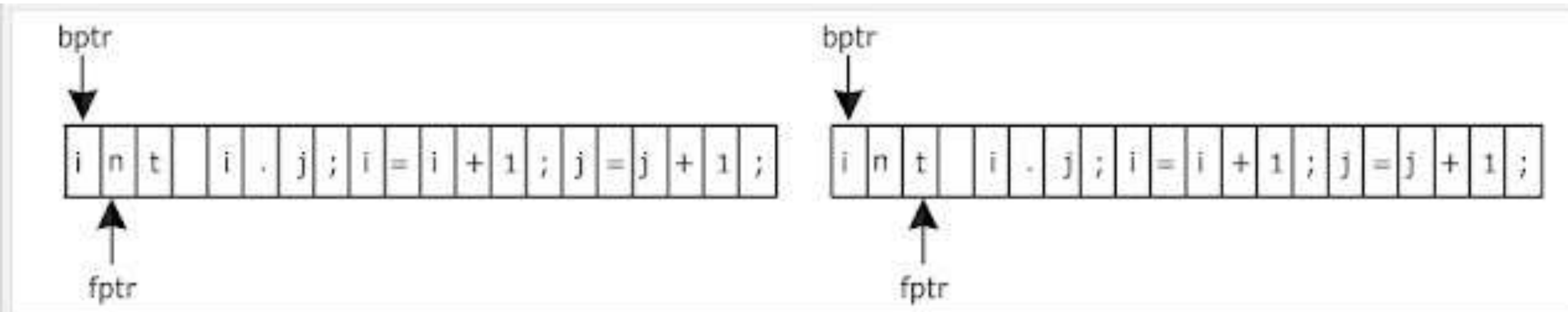
- Two buffer scheme (Initial Configuration)



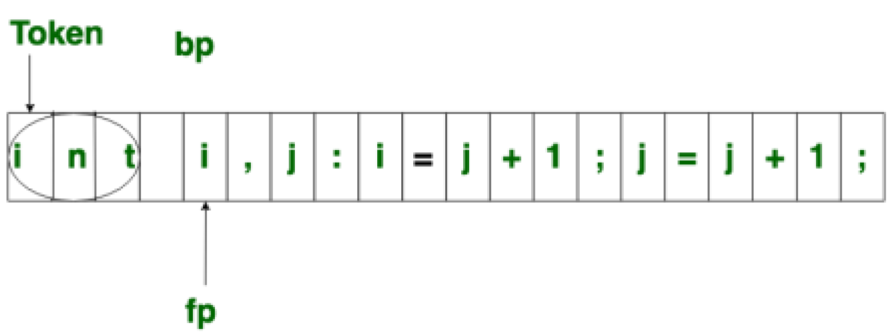
Initial Configuration

Input Buffering

- Two buffer scheme (After reading a token)



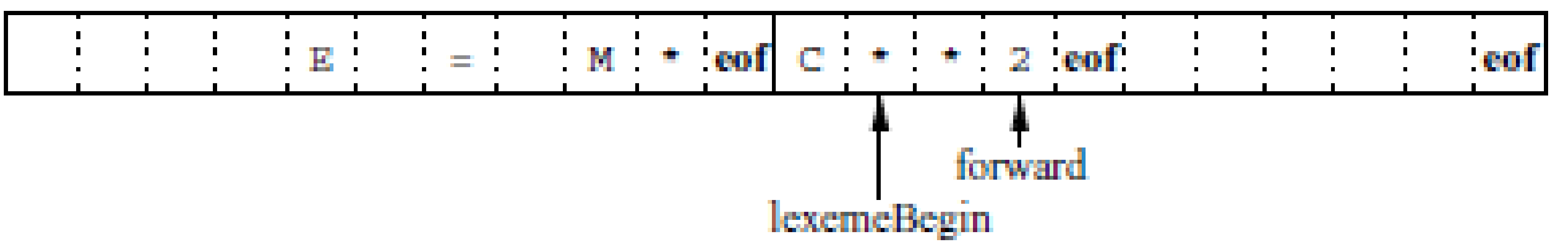
Input Buffering



Input buffering

Input Buffering

- To move the forward pointer
  - Check the end of buffer → reload the other buffer
  - Next character to read
- Combine →Sentinels (eof)
  - eof → end of entire input
  - eof → end of buffer



Input Buffering

<u>Lookahead code with sentinels:</u>

```
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;
}
```

Specification of Token

Specification of Tokens

Regular expressions are an important notation for specifying lexeme patterns

An **alphabet** is a finite set of symbols.

- Typical example of symbols are letters, digits and punctuation etc.
- The set {0, 1} is the binary alphabet.

A **string** over an alphabet is a finite sequence of symbols drawn from that alphabet.

- The length is string s is denoted as |s|
- Empty string is denoted by ε

Prefix: ban, banana, ε, etc are the prefixes of banana
Suffix: nana, banana, ε, etc are suffixes of banana

Kleene or **closure** of a language L, denoted by L*.

- L*: concatenation of L zero or more times
- L^0: concatenation of L zero times
- L^+: concatenation of L one or more times

Specification of Token

Kleene closure

Let: $L = \{ a, bc \}$

L^* denotes "zero or more concatenations of" L

Example:

$L^0 = \{ \varepsilon \}$

$L^1 = L = \{ a, bc \}$

$L^2 = LL = \{ aa, abc, bca, bcbc \}$

$L^3 = LLL = \{ aaa, aabc, abca, abcbc, bcaa, bcabc, bcbca, bcbcbc \}$

...etc...

$L^N = L^{N-1}L = LL^{N-1}$

The "Kleene Closure" of a language:

$$\sum_{i=0}^{\infty} a^i = a^0 \cup a^1 \cup a^2 \cup \ldots$$

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$$

Example:

$$L^* = \{ \underbrace{\varepsilon}_{L^0}, \underbrace{a, bc}_{L^1}, \underbrace{aa, abc, bca, bcbc}_{L^2}, \underbrace{aaa, aabc, abca, abcbc, \ldots}_{L^3} \}$$

Specification of Token

Example

Let: L = { **a**, **b**, **c**, ..., **z** }
D = { **0**, **1**, **2**, ..., **9** }

D^+ = *"The set of strings with one or more digits"*

$L \cup D$ = *"The set of all letters and digits (alphanumeric characters)"*

LD = *"The set of strings consisting of a letter followed by a digit"*

L^* = *"The set of all strings of letters, including ε, the empty string"*

$(L \cup D)^*$ = *"Sequences of zero or more letters and digits"*

$L ((L \cup D)^*)$ = *"Set of strings that start with a letter, followed by zero or more letters and digits."*

Specification of Token

Rules for specifying Regular Expressions

Regular expressions over alphabet Σ

1. ε is a regular expression that denotes $\{\varepsilon\}$.
2. If **a** is a symbol (i.e., if **a** $\in \Sigma$), then **a** is a regular expression that denotes $\{a\}$.
3. Suppose r and s are regular expressions denoting the languages L(r) and L(s). Then
   a) (r) | (s) is a regular expression denoting L(r) U L(s).
   b) (r)(s) is a regular expression denoting L(r)L(s).
   c) $(r)^*$ is a regular expression denoting $(L(r))^*$.
   d) (r) is a regular expression denoting L(r).

Specification of Token

How to "Parse" Regular Expressions

- **Precedence:**
  - * has highest precedence.
  - Concatenation as middle precedence.
  - | has lowest precedence.
  - Use parentheses to override these rules.
- Examples:
  - **a b* = a (b*)**
    - If you want **(a b)*** you must use parentheses.
  - **a | b c = a | (b c)**
    - If you want **(a | b) c** you must use parentheses.
- Concatenation and | are associative.
  - **(a b) c = a (b c) = a b c**
  - **(a | b) | c = a | (b | c) = a | b | c**
- Example:
  - **b d | e f * | g a = (b d) | (e (f *)) | (g a)**

Specification of Token

Example

- Let $\Sigma = \{a, b\}$
  - The regular expression a | b denotes the set {a, b}
  - The regular expression (a|b)(a|b) denotes {aa, ab, ba, bb}
  - The regular expression a^* denotes the set of all strings of zero or more a's. i.e., {ε, a, aa, aaa, ….. }
  - The regular expression $(a|b)^*$ denotes the set containing zero or more instances of an a or b.
  - The regular expression a|a*b denotes the set containing the string a and all strings consisting of zero or more a's followed by one b.

Regular Definition

$$\begin{aligned} letter\_ &\rightarrow \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \_ \\ digit &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\ id &\rightarrow letter\_ \, (\, letter\_ \mid digit \,)^{*} \end{aligned}$$