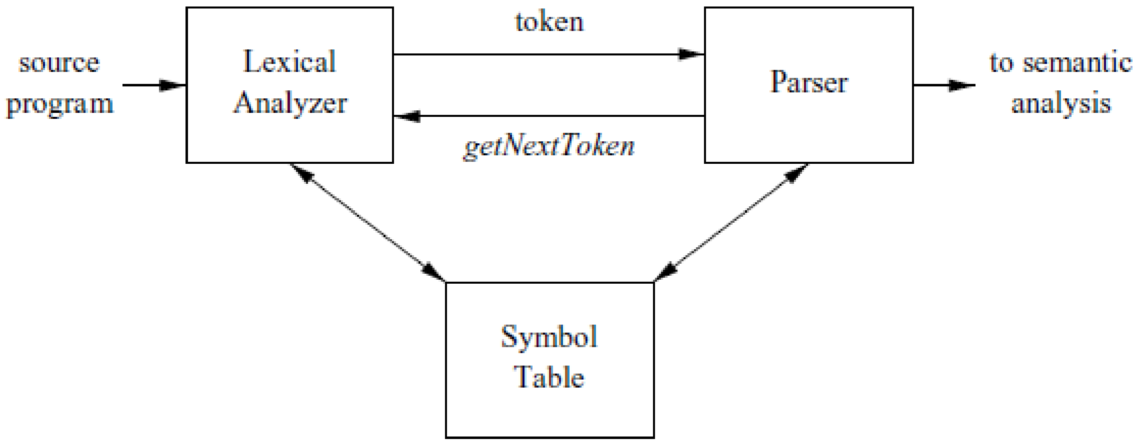
CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

- **Input Buffering**
  - **Buffer Pairs**
  - **Sentinels**
- **Specification of Tokens**
  - **String and Languages**
  - **Operations on Languages**
  - **Regular Expressions (TBD in Part3)**

Input Buffering

- **Before discussing the problem of recognizing lexemes in the input, let us examine some ways that the simple but important task of reading the source program can be speeded**
- **Still remember this?** (LA) produce as output a sequence of tokens for each lexeme in the source program



Input Buffering

- Why we need the buffering?
  - We read the input streams into the buffer, so…
  - To determine the end of an identifier normally requires reading the first whitespace or punctuation character after it.
  - Also just reading > does not determine the lexeme as it could also be >=.
  - When you determine the current lexeme, the characters you read beyond it may need to be read again to determine the next lexeme.

Buffer Pairs

- The book just introduced a “2 size of N” buffers
- eof means, if fewer than N characters remain in the input file, then a special character, represented by eof marks the end of the source file
- By shifting the 2 pointers, Pointer lexemeBegin and Pointer forward to recognize a lexeme
- The “forward” has to “rewind” to recognize a single lexeme, in this example.

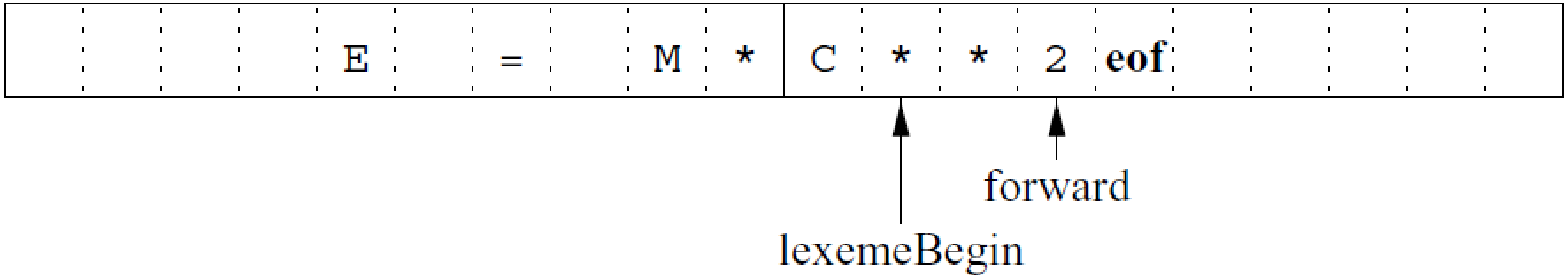


Figure 3.3: Using a pair of input buffers

Buffer Pairs

- If the “forward” pointer is reaching the eof, that means this time of the reading is coming to an end.
- In this example, if the “forward” reads the “eof”, we must **reload the previous one,** and move the “forward” to the beginning of the newly loaded buffer.

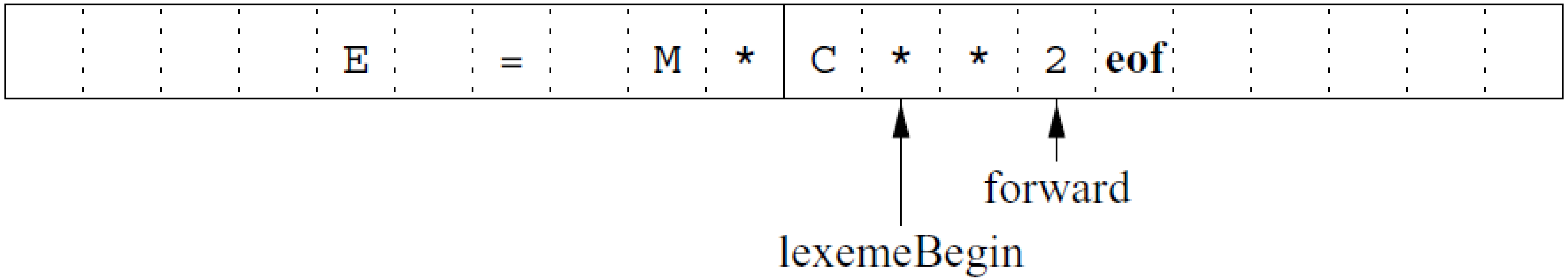


Figure 3.3: Using a pair of input buffers

Sentinels

- The sentinel is a special character that cannot be part of the source program, and a natural choice is the character "eof".
- It is a useful programming improvement to combine testing for the end of a buffer with determining the character read.
- The book is saying a high-level idea to manually put lots of "eof", indicating an input is coming to an end
- Then? We only need to scan and recognize lexemes between "eofs"
- They put an additional

"eof" to, the end of buffer.

(saying, this is the buffer end)

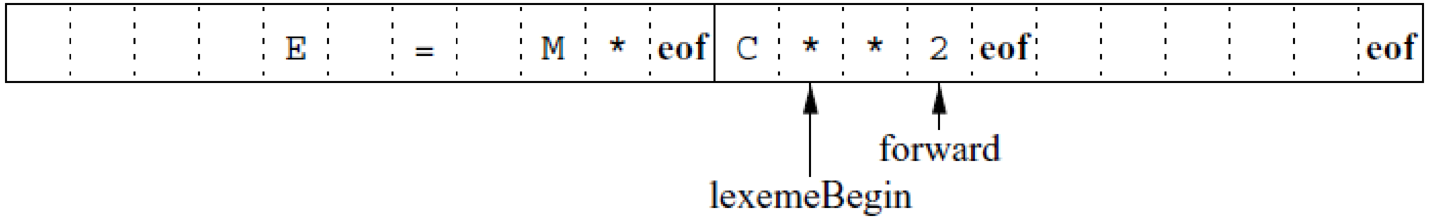


Figure 3.4: Sentinels at the end of each buffer

Sentinels

- Here is an algorithm from the book

```
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;
    Cases for the other characters
}
```

Specification of Tokens

- Regular expressions are an important notation for specifying lexeme patterns.
- Even though they cannot express all possible patterns, they are very effective in specifying those types of patterns that we actually need for tokens.
- In this section we will study the formal notation for regular expressions.
- In Section 3.5, we shall see how these **expressions** are used in a **lexical-analyzer (LA) generator**
- Section 3.7 shows how to **build the LA** by **converting regular expressions** to **automata** that perform the recognition of the specified tokens

Strings and Languages

- Ch3.3.1, this one, strings and languages, we had tons of definitions.

  Definition1: An *alphabet* is a finite set of symbols. It could be {0, 1}, the binary alphabet, the ASCII collection or the Unicode.

  Definition2: A string over an alphabet is a finite sequence of symbols drawn from that alphabet. |s| is used to denote the length of string. The empty string is usually denoted as “epsilon”, ε

  Definition3: A language over an alphabet is a countable set of strings over the alphabet. For example, an English sentence. Some of the cases like empty sets of {ε} are still in the definition. Very broad.

  Definition4: The *concatenation* of strings s and t is the string formed by appending the string t to s, denoted “st”

Strings and Languages

- A small example:
  - We “define” an language: “exponentiation”
  - s^0 → ε
  - For all i > 0, s^i → s^(i-1) s
  - Since εs = s, it follows that:
    - s^1=s, then
    - s^2=ss,
    - s^3=sss, and so on.

Operations on Languages

- In lexical analysis, the most important operations on languages are **union**, **concatenation**, and **closure,** which are defined formally in Fig. 3.6.
- The **concatenation** of languages is all strings formed by taking a string from the **first** language and a string from the **second** language**, in all possible ways,** and concatenating them.
- Definitions again!? The **(Kleene) closure** of a language L, denoted L^*, is the set of strings you get by concatenating L **zero or more times**.
- So, L^0 means the concatenation of L zero times or is called {ε}
- L^i is L^i L
- Positive closure: L^+, the same as {L^*}– {L^0}

Operations on Languages

Operation	Definition and Notation
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

Figure 3.6: Definitions of operations on languages

Operations on Languages

Example 3.3: Let L be the set of letters $\{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}, \texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$ and let D be the set of digits $\{\texttt{0}, \texttt{1}, \ldots \texttt{9}\}$.

languages that can be constructed from languages L and D

1. $L \cup D$ is the set of letters and digits — strictly speaking the language with 62 strings of length one, each of which strings is either one letter or one digit.
2. LD is the set of 520 strings of length two, each consisting of one letter followed by one digit.
3. L^4 is the set of all 4-letter strings.
4. L^* is the set of all strings of letters, including ϵ, the empty string.
5. $L(L \cup D)^*$ is the set of all strings of letters and digits beginning with a letter.
6. D^+ is the set of all strings of one or more digits.

Regular Expressions

- Important (Covered in the Part3)
  - Potential for 1 time of your homework
  - Just some practices in the Linux / Unit environments
  - Try to find out strings / keywards in the files like hacker’s way