Compilers

by
Marwa Yusuf

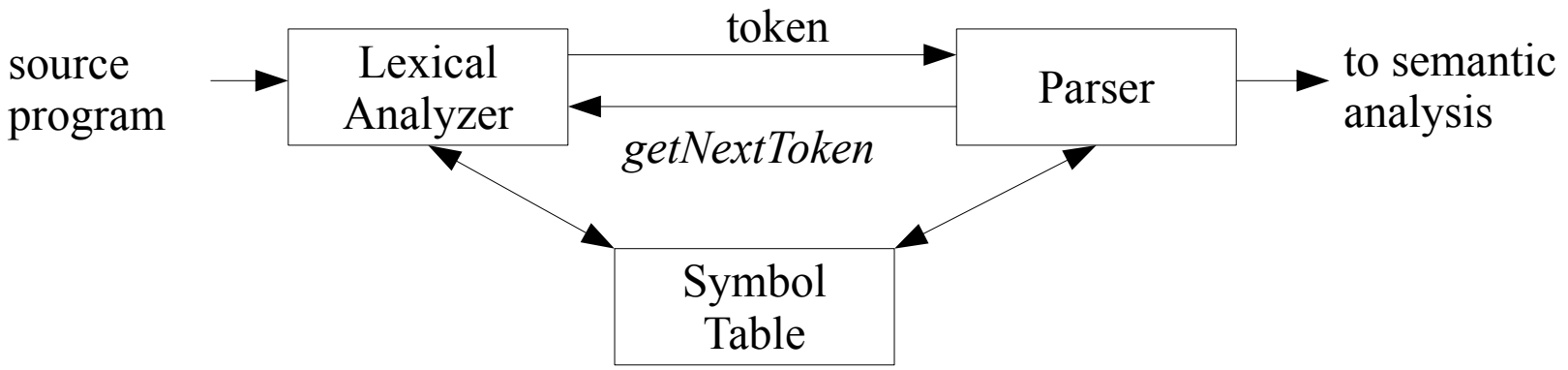
Lecture 2
Tues. 23-3-2021

Chapter 3 (3.1 to 3.2)

Lexical Analysis

The Role of the Lexical Analyzer

- Read input characters, group into lexemes, produce a sequence of tokens
- May insert into symbol table (identifier)



- Strip out comments and whitespace
- Correlating error messages with source (line No.)
- Expansion of macro-preprocessor

Scanning + lexical analysis

Why separate lexical analysis from parsing?

- Simplicity of design (comments and whitespace)
- Compiler efficiency (specialized techniques, buffering)
- Portability (Input-device-specific peculiarities)

Tokens, patterns and lexemes

- **Token**: name + optional attribute value
- **Pattern**: form description of the token lexemes
- **Lexeme**: sequence of characters that matches the token pattern (instance of token)
- **Ex:** `printf("Total = %d\n", score);`

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters **i, f**	**if**
else	characters **e, l, s, e**	**else**
comparison	< or > or <= or >= or == or !=	<=, !=
id	letter followed by letters and digits	**pi, score, D2**
number	any numeric constant	**3.14159, 0, 6.02e23**
literal	anything but ", surrounded by "'s	**"core dumped"**

Tokens, patterns and lexemes

- In many languages:
  1) 1 token for each keyword
  2) Tokens for operators (individually or grouped)
  3) 1 token for identifiers
  4) 1 or more tokens for constants (numbers, literals)
  5) Tokens for each punctuation (parenthesis, commas ...etc)

Attributes for Tokens

- If token pattern represents more than one lexeme, then info about the specific lexeme must be stored and passed (like id.lexeme)
- Token name affects parsing, attribute affects translation.
- Usually one attribute (that may refer to a structure like a symbol table record)
- Ex: E = M * C ** 2

Difficulty in identifying tokens:
In fixed format in Fortran 90
DO 5 I = 1.25
DO 5 I = 1,25

Lexical Errors

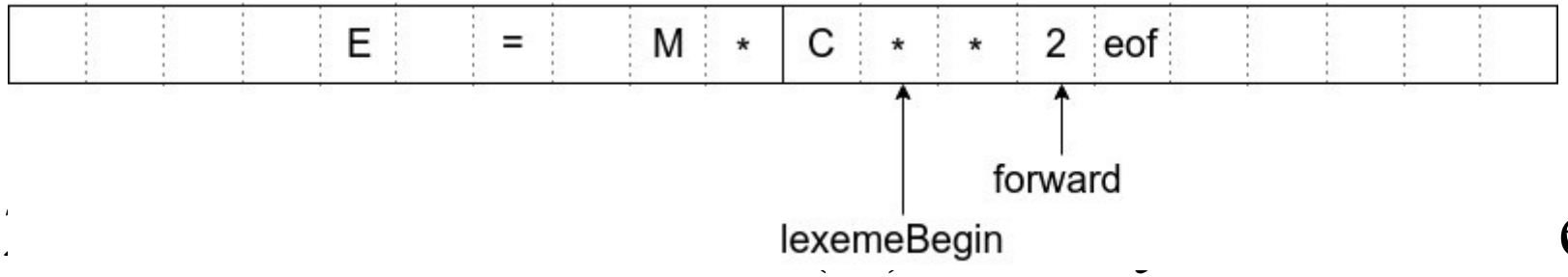
- Ex: fi (a == f(x)) …
  - fi is a valid identifier, lexical analyzer will pass it
- If no token can be formed:
  - panic mode: delete until token can be formed
  - other techniques:
    - delete 1 char
    - insert a missing char
    - replace a char by another
    - transpose 2 adjacent characters
  - A single transformation to proceed.
  - Find the min no. of transformations to transform input programs into a valid seq. of lexemes, very expensive to be practical.

Input Buffering

- Many times, at least one more character is needed to decide the end of lexeme:
  - id: must see a character that is not a digit nor a letter
  - <, =, >: may be <=, ==, >-

Buffer Pairs

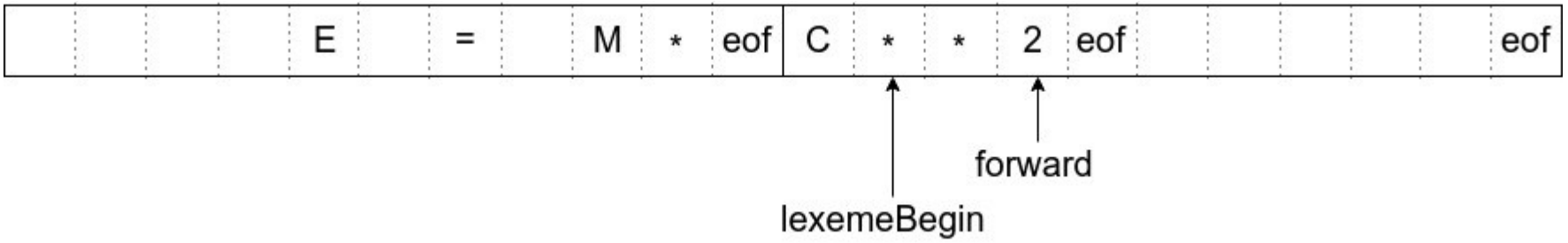
- Processing characters is expensive and programs have large number of characters.
- Hence, specialized buffering technique is developed.



- e (e.g. 4096 bytes), alternately reloaded.
- Condition: lexeme length + lookahead <= N (to not overwrite the current lexeme)

Sentinels

- 2 tests for each *forward*: end of buffer (reload the other and move to its beginning) and the actual character (may be multiway branch)
- Add sentinel character at the end of buffer (eof)



Lookahead code

```
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 used to specify lexeme patterns.

Strings and Languages

- **Symbols**: digits, letters, punctuation.
- ***Alphabet*** Σ: any finite set of symbols.
  - {0, 1} binary alphabet.
  - ASCII
  - Unicode: 100,000 symbols.
- ***String*** (over an alphabet): finite seq. of symbols drawn from alphabet.
- **|s|**: length of string, number of occurrences of symbols.
- ***Empty string*** ε: the zero length string.

Strings and Languages

- ***Language***: any countable set of strings over some fixed alphabet, meaning is not a condition.
  - φ, empty set, {ε} are languages.
  - All syntactically well-formed c programs.
  - All grammatically correct English sentences.
- ***Concatenation*** of *x* and *y* (*xy*): appending *y* to *x*.
  - $\varepsilon s = s\varepsilon = s$
- ***Exponentiation***:
  - $s^0 = \varepsilon$
  - $s^i = s^{i-1}s$
  - $s^1 = s,\ s^2 = ss,\ s^3 = sss\ \ldots$

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	$L^+ = \cup_{i=1}^{\infty} L^i$

- L = {A, B, …, Z, a, b, …, z} – D = {0, 1, …, 9}
  - Alphabet or a language of one letter

  1) L ∪ D
  2) LD
  3) L^4
  4) L^*
  5) $L(L \cup D)^*$
  6) D^+