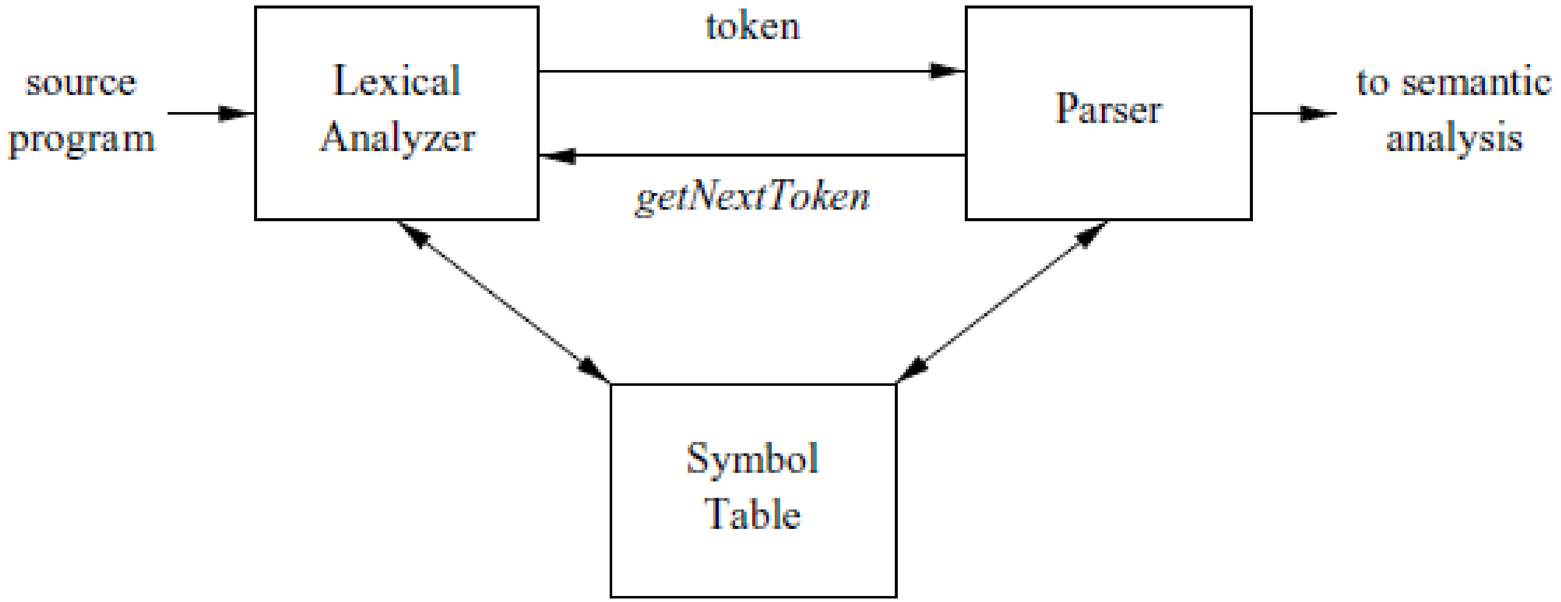
CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

- **The Role of the Lexical Analyzer (3.1)**
  - **LA vs. Parsing**
  - **Tokens, Patterns, and Lexemes**
  - **Attributes for Tokens**
  - **Lexical Errors**
- **Input Buffering (TBD, in Part2)**

The Role of the Lexical Analyzer

- The lexer is called by the parser when the latter is ready to process another token.
- lexical analyzer, read the input characters of the source program, group them into lexemes



The Role of the Lexical Analyzer

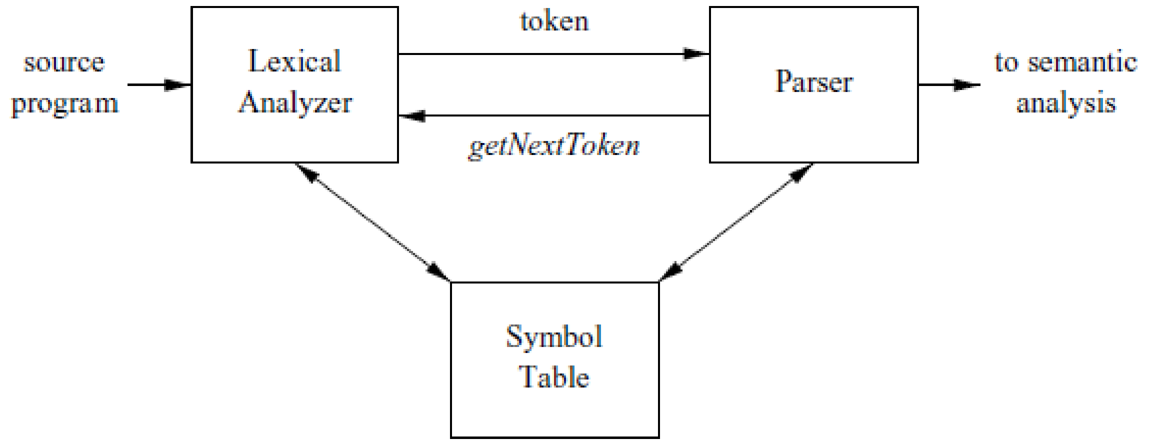
- Also, lexical analyzer (LA) produce as output a sequence of tokens for each lexeme in the source program
- The stream of tokens is sent to the parser for syntax analysis
- LA needs to interact with symbol table
- When, LA discovers a lexeme constituting an ID (identifier), it needs to enter that lexeme into the symbol table
- Commonly, the interaction is

implemented by having the parser

call the LA

- getNextToken(). Hey, please give me next

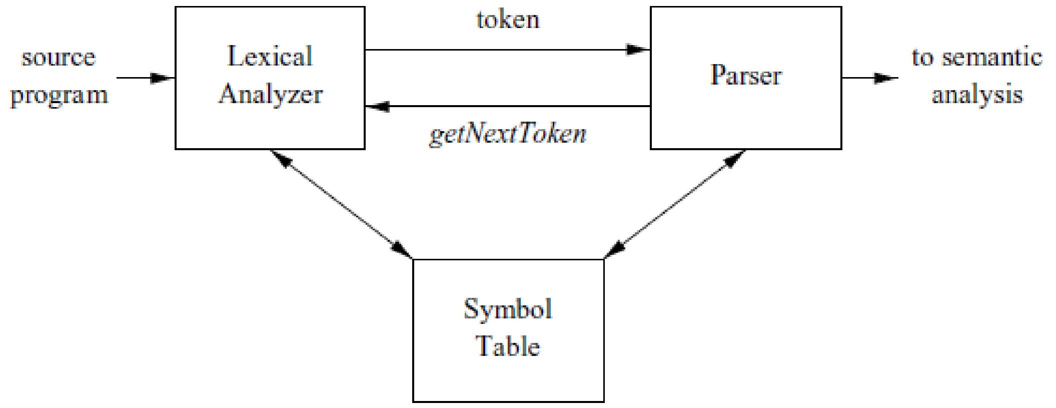
token



The Role of the Lexical Analyzer

- For this call, getNextToken(), will push the LA to read characters from input until a next lexeme is identified then produce a token. Send it to parser
- LA also strips out comments and whitespace (blank, newline, tab, …, etc.)
- LA can be divided into 2 parts
  - Scanning: purely read through the source
  - The part to produce tokens,

  from the output of the scanner



LA vs. Parsing

- A couple of reasons why we need to separate LA from Parsing
  - SW engineering idea
    - One can expect the white spaces, tabs is already removed
    - Efficiency. In the LA, Lexical Errors specialized buffering techniques for reading input characters can speed up the compiler significantly.

Tokens, Patterns, and Lexemes

- Token
  - A token is a pair consisting of a token name and an optional attribute value.
  - The token name is an abstract symbol representing a kind of lexical unit.
  - The token names are the input symbols that the parser processes.
- Pattern
  - A description of the form that the lexemes of a token may take.
  - i.e. identifier (ID) has a pattern. Keyward has a patter.
- Lexeme
  - A sequence of characters that matches the pattern for a token and is identified by the LA as an instance of a token

Tokens, Patterns, and Lexemes

- Example: How the **lexeme** matches the **pattern** and form a **token**

```
    printf("Total = %d\n", score);
```

both `printf` and `score` are lexemes matching the pattern for token **id**, and `"Total = %d\n"` is a lexeme matching **literal**. □

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"`

Figure 3.2: Examples of tokens

Tokens, Patterns, and Lexemes

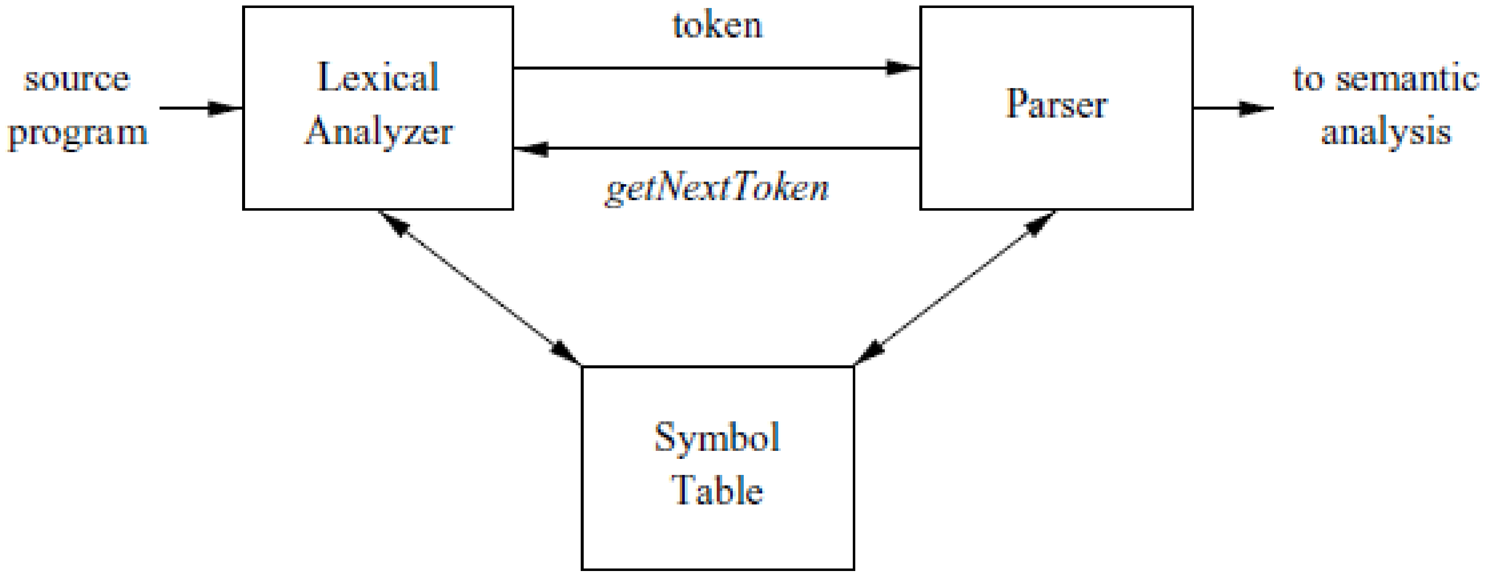
- Some rules:
  - One token for each keyword
  - One token representing all identifiers (variables)
  - Tokens for each punctuation symbol, such as left and right parentheses, comma, and semicolon

Attributes for Tokens

- When more than one lexeme can match a pattern, the lexical analyzer must provide the subsequent compiler phases additional information about the particular lexeme that matched
  - i.e. If there is a token "**number**" matches both 0 and 1, but it is extremely important for the code generator to know which lexeme was found in the source program? 0 or 1?

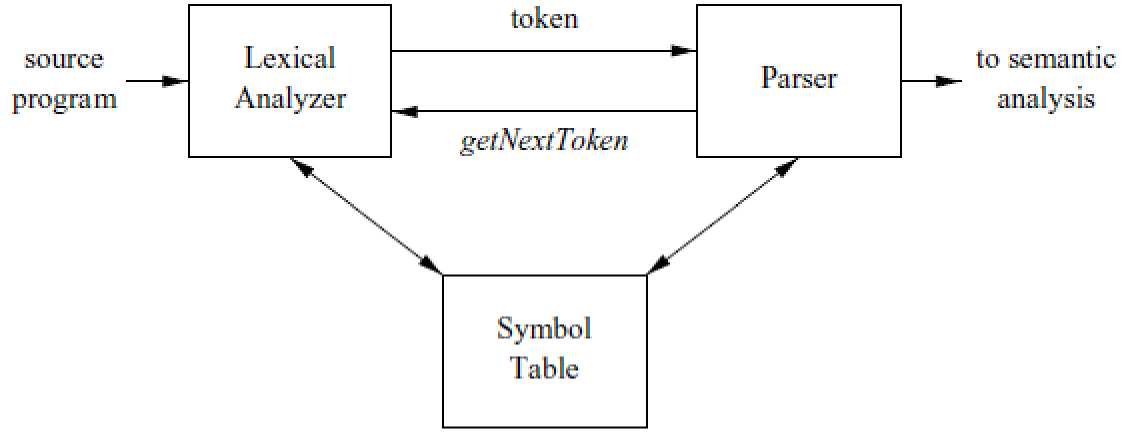
Attributes for Tokens

- In many cases, the LA returns to the parser not only a token name, but an “attribute value” that describes the lexeme represented by the token
- The token name can influence the parsing decision
- Attribute value influences translation of tokens after the parse.
- We shall assume that tokens have at most one associated attribute.



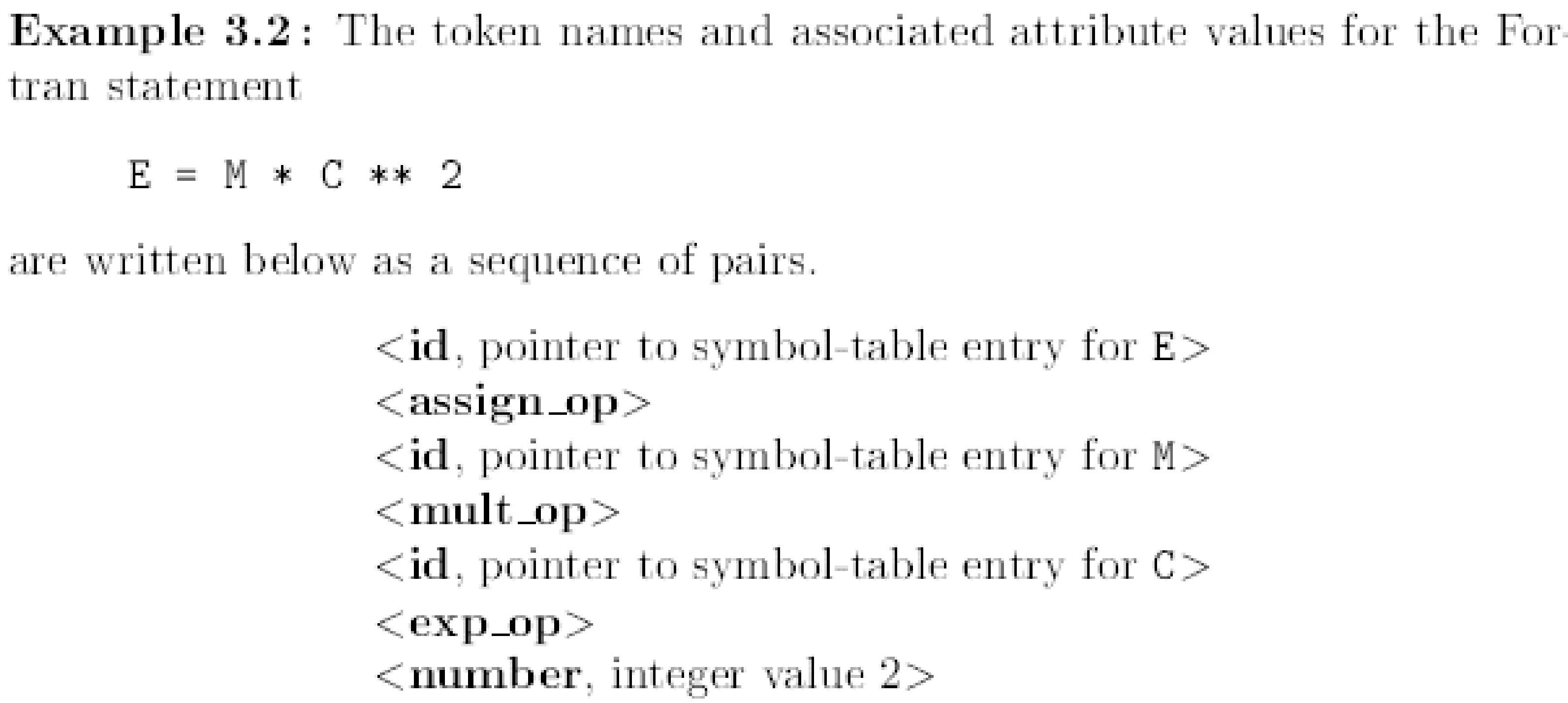
Attributes for Tokens

- The most important example is the token id, where we need to associate with the token a great deal of information.
- Information about an ID
  - e.g., its lexeme, its type, and the location at which it is first found (in case an error message about that id must be issued) --- is kept in the symbol table.
  - Thus, the appropriate attribute value for an ID is a pointer to the symbol-table entry for that identifier.



Attributes for Tokens

- An example for tokens
  - Note that In certain pairs, especially operators, punctuation, and keywords, there is no need for an attribute value



Example 3.2: The token names and associated attribute values for the For-
tran statement

```
E = M * C ** 2
```

are written below as a sequence of pairs.

<**id**, pointer to symbol-table entry for E>
<**assign_op**>
<**id**, pointer to symbol-table entry for M>
<**mult_op**>
<**id**, pointer to symbol-table entry for C>
<**exp_op**>
<**number**, integer value 2>

Lexical Errors

- For instance, if the string “fi” is encountered for the 1st time in a C program in the context:
  - fi (a == f(x)) ...
  - LA cannot tell whether “fi” is a misspelling of the keyword if or an undeclared function ID.
  - Since fi is a valid lexeme for the token id for LA, the LA must return the token ID to the parser and let some other phase of the compiler -- probably the parser itself in this case to handle an error --- due to transposition of the letters

Lexical Errors

- Suppose, when, there is a case happens.
- The LA is unable to proceed because none of the patterns for tokens matches any prefix the remaining input.
  - The simplest recovery strategy is “panic mode” recovery.
  - Deleting successive characters from the remaining input, until the LA can find a well-formed token at the beginning of what input is left.
  - This recovery technique may confuse the parser, but in an interactive computing environment it may be quite adequate.

Lexical Errors

- Other possible error-recovery actions are:
  - Delete one character from the remaining input.
  - Insert a missing character into the remaining input.
  - Replace a character by another character.
  - Transpose two adjacent characters
- Those attempts are about trying to repair the input.
- The strategies make sense because most of the errors involve a single character

Input Buffering

- **(TBD, in Part2)**
- **I was thinking about your HW2 which is about parsing HTML documents by some programming languages**
- **A small programming practice. No harms ☺**