CMP4060 Languages and Compilers
Lexical Analysis – Part1

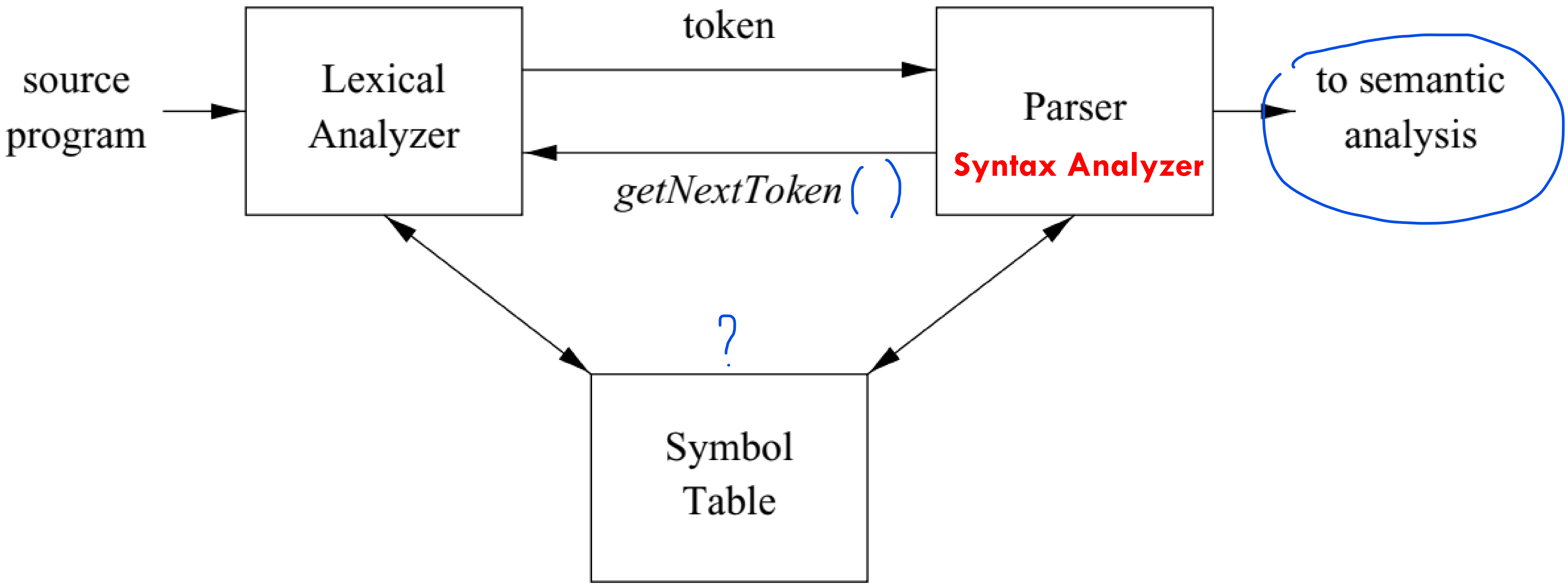
Ayman AboElHassan, PhD
Assistant Professor
ayman.abo.elmaaty@eng.cu.edu.eg

Course Outline

- Introduction to Compilers
- Lexical Analysis: Regular Grammars
- Lexical Implementation: Finite Automata
- Syntax Analysis: Context-Free Grammars
- Parser Implementation: Top-Down Parsers
- Parser Implementation: Bottom-Up Parsers
- Semantic Analysis
- Code Generation
- Code Optimization

Lexical Analysis Role

Role of Lexical Analysis



Role of Lexical Analysis

?

Why Separate Lexical & Syntax Analyzers?

1. Simple Compiler Design
   - No Lexical ➔ Parser needs to deal with comments and whitespaces
   - Cleaner overall language design

Role of Lexical Analysis

Why Separate Lexical & Syntax Analyzers?

1. Simple Compiler Design
2. Improve Compiler Efficiency
   - Apply specialized techniques (i.e.: read input buffering)

Role of Lexical Analysis

Why Separate Lexical & Syntax Analyzers?

1. Simple Compiler Design
2. Improve Compiler Efficiency
3. Portability

Role of Lexical Analysis

Input: a string of characters

- Example

\tif(i==j)\n\t\tz = 0;\n\telse\n\t\tz = 1; ?

Output: a set of substrings “Tokens”

- Example

<if> <i> <==> <j> <z> <=> <0> <;> <else> <z> <=> <1> <;>

```
if i == int
    z = 0
else
    z = 1
```

Designing Lexical Analyzer

Designing Lexical Analyzer (Step1)

Define a finite set of tokens

- Tokens describe all items of interest
- Language dependent
  - if then else?
  - var?
  - tab space?

Token Types

Identifiers: x y11 elsen_i00

Keywords: if else while for return

Constants

- Integer: 2 1000 -500 0x777
- Float-point: 2.0 0.00020 .02 1. 1e5 0.5-10
- String: “x” “X = %d\nY = %d”
- Character: ‘c’

Symbols: + * { } ++ < [ ] > =

Whitespaces

- Typically recognized & discarded
- Comments, Spaces “ ”, and Format characters “\n” “\t”

Designing Lexical Analyzer (Step2)

Describe which patterns belong to each token

- A Token could have multiple patterns

Example

- Keyword “if” ➔ a single pattern “**if**”
- Integer ➔ Multiple pattern

  Pattern1: **[0-9]+**

  Pattern2: **0b[0-1]+**
- Floating point ➔ Multiple patterns

  Pattern1: **[0-9]+.[0-9]+**

  Pattern2: **[0-9]+e[0-9]+**

Token vs Pattern vs Lexeme

Token

- A pair of "**Token name**" and "Optional **Attribute value**"
- Token name is an abstract symbol representing kind (i.e.: specific keyword, identifier notation)

Pattern

- A description of the form that the lexemes of a token may take.

Lexeme

- A sequence of characters in the source program that matches the pattern of a token

Token vs Pattern vs Lexeme

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"`

Token vs Pattern vs Lexeme

E = M * C ** 2

Token vs Pattern vs Lexeme

E = M * C ** 2

<**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>

Implementation

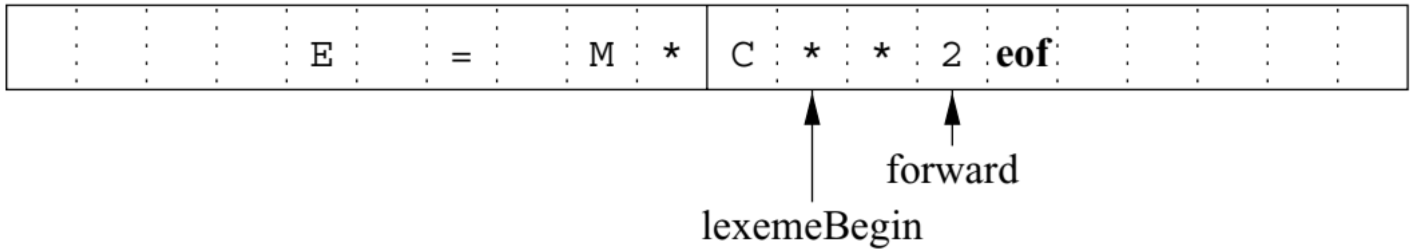
Lexical Analyzer Implementation

1. Recognize substrings corresponding to tokens
2. Return the “lexeme” value of the token
3. Discard “uninteresting” tokens

Issues Handled During Implementation

Lookahead

- Determine the start and end of a lexeme
- Example
  - i vs if vs if_flag
  - = vs ==
- Use a pair of pointers



Issues Handled During Implementation

Ambiguities

- Keywords that are not reserved
  - i.e.: “max = 5” vs “max(1,5)”
- Scope start/end
- User-defined types
  - i.e.: “Class car”

Issues Handled During Implementation

Lexical Errors

- Example: fi (a == f(x))
  - Variable called "fi" vs misspelling of the keyword "if"

Solution 1: Let the parser handle it ✓

- Mark "fi" as an identifier token
- Pass it to the syntax analyzer "parser"

Issues Handled During Implementation

Lexical Errors

- Example: fi (a == f(x))
  - Variable called "fi" vs misspelling of the keyword "if"

Solution 2: Test error recovery actions

- If the input string doesn't match any pattern
- Test multiple transformations
  - Delete 1 character
  - Insert a missing character
  - Replace a character with another
  - Transpose 2 adjacent characters

dp

Language

What is a Language?

Alphabet Σ

- Any finite set of symbols (Letters, digits, punctuations)

Example

- {0, 1} Binary alphabet
- ASCII

What is a Language?

String

- A finite sequence of symbols drawn from that alphabet
- A word or sentence ➔ string
- $|s|$ ➔ length of a string
  - The number of occurrences of symbols in the string s
- ε is the empty string with $|s| = 0$

What is a Language?

String

- A finite sequence of symbols drawn from that alphabet
- A word or sentence ➔ string
- $|s|$ ➔ length of a string
  - The number of occurrences of symbols in the string s
- ε is the empty string with $|s| = 0$

String Operations

- Concatenation is the product of multiple strings ➔ $x * y$?
- Repetition is the exponentiation of strings ➔ $s^2 = s * s$ ✓

What is a Language?

Formal Language Σ^*

- The set of all possible strings that can be generated from a given alphabet

What is a Language?

Language L

- Set of string characters drawn from alphabet Σ
- A subset of the formal language Σ^*
- Operations can be done 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	$L^+ = \cup_{i=1}^{\infty} L^i$

What is a Language?

Grammer?

Regular Expressions

Regular Expressions

Several formalisms for specifying tokens

Regular expressions are the most popular

- Simple and useful theory
- Easy to understand
- Efficient implementations

Regular Expressions

Algebraic notation for describing sets of strings

Definition of regular expressions over Σ

- Rules that define exactly the set of words that are valid tokens in a language

Atomic Regular Expressions

Single character

- ‘c’ = {“c”}

Epsilon

- $\varepsilon = \{""\}$

Notation

- Italics denote symbols
- Bold denotes regular expressions

Compound Regular Expressions

Union

- $A + B = \{s \mid s \in A \; or \; s \in B\}$

Concatenation

- $AB = \{ab \mid a \in A \; and \; b \in B\}$ ✓

Iteration

- $A^* = U_{i \geq 0} A^i$, where A^i is A concatenated i times

?

Regular Languages

Languages can be defined by a regular expressions

- $L(\varepsilon) = \{""\}$
- $L('c') = \{"c"\}$
- $L(A + B) = L(A) \cup L(B)$
- $L(AB) = \{ab \mid a \in L(A)\ and\ b \in L(B)\}$
- $L(A^*) = \bigcup_{i \geq 0} L(A^i)$

?

Regular Languages

Languages can be defined by a regular expressions

- A = L(A)
- Regular expression **A** matches the set of strings belong to language **L(A)**

Example: Integers

Non-empty string of digits

Regular Definition?

Example: Integers

Non-empty string of digits

Regular Definition

- digit = ‘0’ + ‘1’ + ‘2’ + ‘3’ + ‘4’ + ‘5’ + ‘6 + ‘7’ + ‘8’ + ‘9’
- Integer = digit digit*

Union

Concat

Iteration

?

Example: Identifiers

String of letters or digits starting with a letter

Regular Definition?

Example: Identifiers

String of letters or digits starting with a letter

Regular Definition?

- digit = ‘0’ + ‘1’ + ‘2’ + ‘3’ + ‘4’ + ‘5’ + ‘6 + ‘7’ + ‘8’ + ‘9’
- letter = ‘A’ + ‘B’ + … + ‘Z’ + ‘a’ + ‘b’ + … + ‘z’
- identifier = letter(letter + digit)*

Concat

Union

Iteration

Example: Keyword & Whitespace

Keyword

- Regular expression looks exactly like the keyword
- keyword = "if" + "else" + "then" + "include" + …

Whitespace

- Non-empty sequence of blanks, newlines, and tabs
- whitespace = (' ' + '\n' + '\t')+

Example:

Email?

Phone Numbers?

Extensions to Regular Expressions

Enhance the ability to specify strings

- One or more instances

  $r^* = r^+|\varepsilon$ and $r^+ = rr^* = r^*r$

- Union of regular expressions

  $\mathrm{A} \mid \mathrm{B} \Leftrightarrow \mathrm{A} + \mathrm{B}$

- Optional

  $\mathrm{A} + \varepsilon \Leftrightarrow \mathrm{A?}$

- Range

  'a' + 'b' + … + 'z' ⇔ [abc…z] ⇔ [a-z]

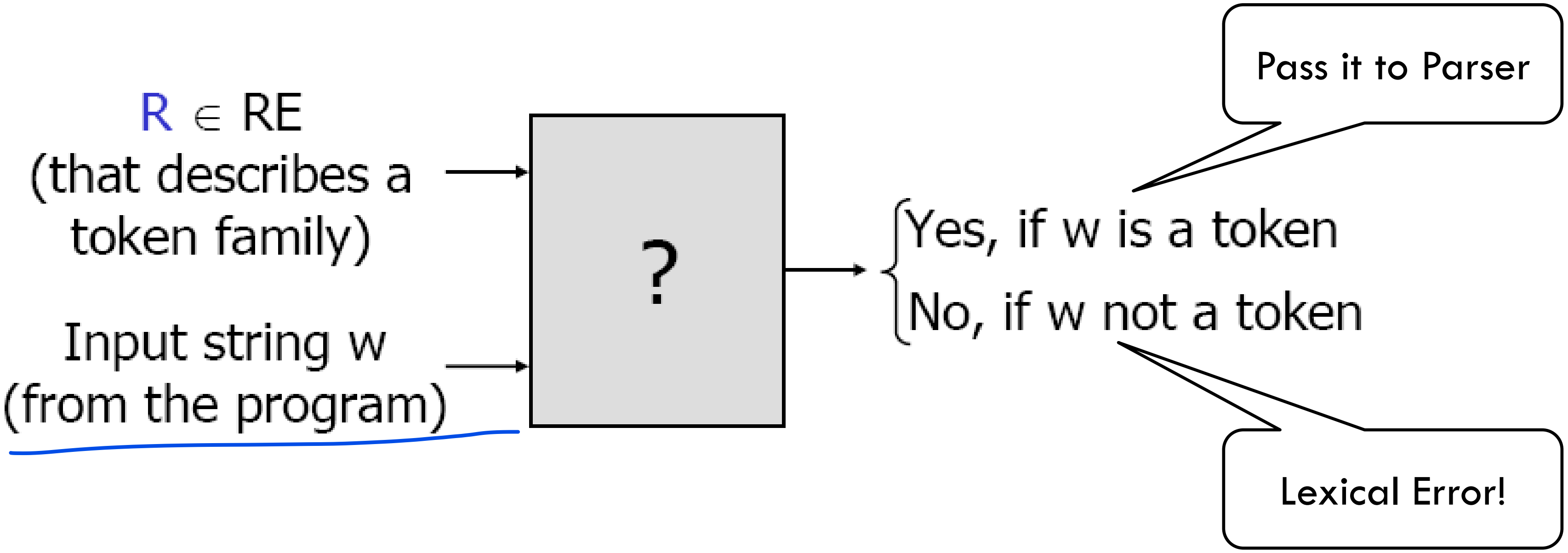
Extensions to Regular Expressions

Regular expressions mapping to languages

- $\mathrm{A} \mid \mathrm{B} = \mathrm{L}(\mathrm{A}) \cup \mathrm{L}(\mathrm{B})$
- Union of both regular expression **A** & **B** matches the union of language sets **L(A)** & **L(B)**

How to use RE in Lexical Analyzer?

Given R ∈ RE and input string w, need a mechanism to determine if w $\in L(R)$



Summary

Regular expressions describe many useful languages

Regular languages are a language specification

- We still need an implementation!

Next time: Given a string s and a regular expression R , How can we decide if:

- s belongs to L(R)

Thank you