Specification of Tokens

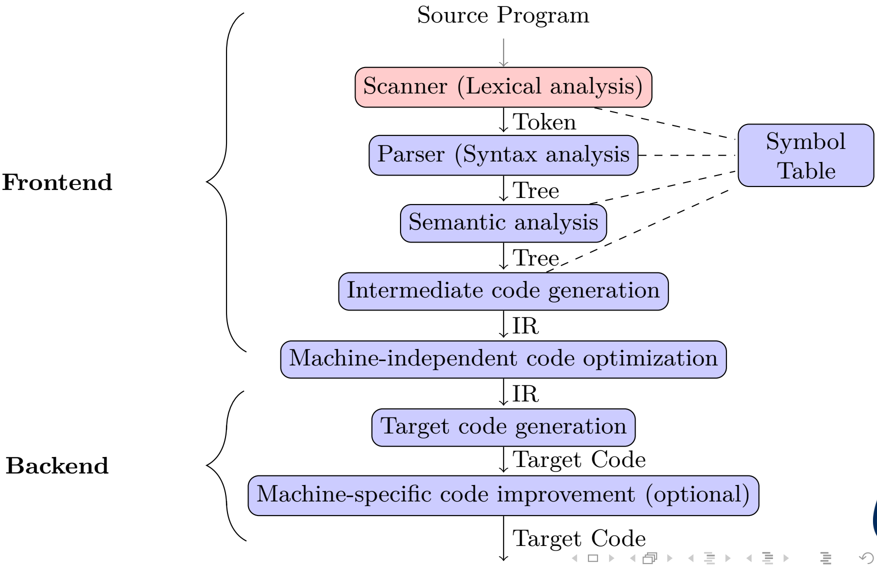
Lecture 2

Section 3.3

Rokan Uddin Faruqui

Associate Professor
Dept of Computer Science and Engineering
University of Chittaong, Bangladesh
Email: *rokan@cu.ac.bd*

The Phases of Compilation



Outline

Lexical Analyzer

```
int x;
float y = 10;
char ch;
```

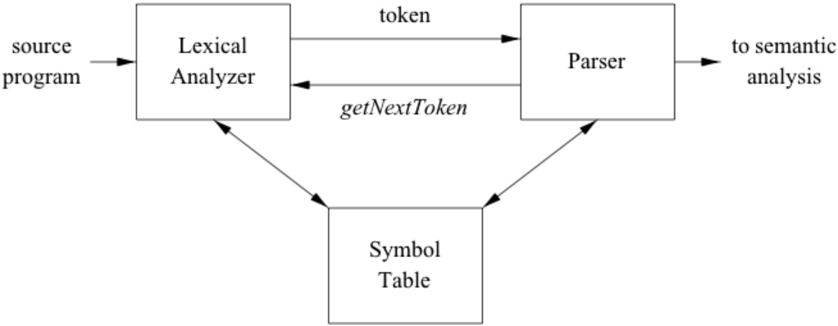


Figure 3.1: Interactions between the lexical analyzer and the parser

Tasks of the Lexer/Scanner

- **read** the input characters of the source program
- **group** them into lexemes
- produce as output a sequence of **tokens** for each lexeme in the source program
- stream of tokens is sent to the parser for syntax analysis
- interact with the symbol table
- inserts the identifiers into symbol table.

Scanning consists of the simple processes that do not require tokenization of the input, such as deletion of comments and compaction of consecutive whitespace characters into one.

Lexical analysis proper is the more complex portion, which produces tokens from the output of the scanner

Scanning Vs. Parsing

Why separating lexical and syntactic analysis?

1. Simplicity of design is the most important consideration
2. Compiler efficiency is improved.
3. Compiler portability is enhanced

Tokens, Patterns, and Lexemes

Token -

- is a pair consisting of a token name and an optional attribute value
- token name is an abstract symbol representing a kind of lexical unit, e.g.,
  - a particular keyword
  - a sequence of input characters denoting an identifier.

Tokens, Patterns, and Lexemes

count

abc123

12asd

d —> [0-9]

l —>[a-zA-Z]

id —> l (l |d) *

Pattern -

- is a description of the form that the lexemes of a token may take.
- In the case of a keyword as a token
  - the pattern is just the sequence of characters that form the keyword

Tokens, Patterns, and Lexemes

Lexeme -

- is a sequence of characters in the source program that matches the pattern for a token
- is identified by the lexical analyzer as an instance of that token.

Example

The stream of characters:

$$position \; = \; initial \; + \; rate \; * \; 60$$

Scanned into list of tokens, one for each lexeme:

$$\mathbf{id}_1 \; = \; \mathbf{id}_2 + \mathbf{id}_3 \; * \; \mathbf{num} \; ;$$

1	position
2	initial
3	rate

Example

x <> y

Lexeme	Token	Lexeme pattern (informal)	Attribute value
position	id_1	identifier string	1
=	<=>	equality symbol	
initial	id_2	identifier string	2
+	<+>	addition symbol	
rate	id_3	identifier string	3
*	<*>	multiplication symbol	
60	<num, 60>	numeric constant	60

Dealing with errors

int abc;

- Panic mode recovery: delete characters from input until a matching pattern is found.
- Insert missing character.
- Replace a character with another.
- Transpose two adjacent characters

Outline

Alphabets and Strings

Definition (Alphabet)

An alphabet is a finite set of symbols. Traditionally, we denote an alphabet by the letter Σ.

Definition (String)

A string is a finite sequence of symbols.

Definition (Empty String)

The empty string, denoted ε, is the string that contains no symbols. The empty string has length 0.

Alphabets and Strings

Example (Alphabets and Strings)

- Examples:
  - The traditional alphabet is

$$\Sigma = \{A, B, C, \ldots, Z\}.$$

  - The binary alphabet is $\Sigma = \{0, 1\}$.
  - For C programs, the alphabet is the set of ASCII characters.

Languages

L = { ww^r| w in Sigma^* } = {0,1,00,11, 0110, 1001,..}

Sigma = { 0, 1}

Sigma^* = {e, 0, 1, 11, 00, 01, 10, ..}

Definition (Language)

A language is a set of (finite) strings over a given alphabet.

- A language can be (and usually is) infinite.
- The set of all even integers over the alphabet $\Sigma = \{0, 1, \ldots, 9\}$ is a language.
- The set of all C programs is a language.

Outline

Operations on Languages

Definition (Language)

Let L, L_1, and L_2 be languages.

- Union:

$$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}.$$

- Concatenation:

4^3= 444

$$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}.$$

- Repeated concatenation:

$$L^n = LLL \cdots L \; n \text{ copies of } L.$$

Operations on Languages

Example (Language) L1^3={111,333,135,}

- Let

$$L_1 = \{1, 3, 5, 7, 9\}$$

and

$$L_2 = \{0, 2, 4, 6, 8\}.$$

Text

- Describe $L_1 \cup L_2$. **01**
- Describe $L_1 L_2$. **={10,12,14,16,18,30,32,34,36,... }**
- Describe $(L_1 \cup L_2)^3$.
- Describe $(L_1 \cup L_2)^* L_2$.

Outline

Regular Expressions

- A regular expression can be used to describe a language.
- Regular expressions may be defined in two parts.
  - The basic part.
  - The recursive part.

Regular Expressions

- The basic part:
  - ε represents the language $\{\varepsilon\}$.
  - $\mathbf{a}$ represents the language $\{a\}$ for every $a \in \Sigma$.
  - Call these languages $L(\varepsilon)$ and $L(\mathbf{a})$, respectively.

Regular Expressions

L(r) L(s)

- The recursive part: Let r and s denote regular expressions.
  - $r \mid s$ represents the language $L(r) \cup L(s)$.
  - rs represents the language $L(r)L(s)$.
  - r^* represents the language $L(r)^*$.
  - (r) represents the language $L(r)$.

Regular Expressions

- In other words
  - $L(r \mid s) = L(r) \cup L(s)$.
  - $L(rs) = L(r)L(s)$.
  - $L(r^*) = L(r)^*$.
  - $L((r)) = L(r)$.

Example

{A}| {B}| ... = {a,b,..,z,A,B,...Z}

Example (Identifiers)

- Identifiers in C++ can be represented by a regular expression.

$$r = \mathtt{A} \mid \mathtt{B} \mid \cdots \mid \mathtt{Z} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{z}$$

$$s = \mathtt{0} \mid \mathtt{1} \mid \cdots \mid \mathtt{9}$$ **={0,1,...9}**

$$t = r(r \mid s)^*$$

t={a,b,c,..,A0, AA, A0ab1..} **1A0A**

Regular Expressions

Definition (Regular definition)

A regular definition of a regular expression is a “grammar” of the form

$$
\begin{aligned}
d_1 &\to r_1 \\
d_2 &\to r_2 \\
&\vdots \\
d_n &\to r_n
\end{aligned}
$$

where each r_i is a regular expression over $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$.

Example

int —> digit+

102, 8889 **digit —> 1**

Example (Identifiers)

- We may now describe C++ identifiers as follows.

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

Regular Expressions

- Note that this definition does not allow recursively defined tokens.
- In other words, d_i cannot be defined in terms of d_i, not even indirectly.

Outline

Extensions of Regular Languages

r * = zero or more

r+ = one or more = rr *

r? = r | e

[a-z] = a|b|c|...|z

[abc] = a|b|c

Definition

We add the following symbols to our regular expressions.

- One or more instances: $r^+ = r\, r^*$.
- Zero or one instance: $r? = r \mid \varepsilon$.
- Character class: $[a_1 a_2 \cdots a_n] = a_1 \mid a_2 \mid \cdots \mid a_n$.

Extensions of Regular Languages

Example (Identifiers)

- Identifiers can be described as

$$
\begin{aligned}
letter &\rightarrow [\texttt{A-Za-z}] \\
digit &\rightarrow [\texttt{0-9}] \\
id &\rightarrow letter\ (letter \mid digit)^*
\end{aligned}
$$

Extensions of Regular Languages

100.899
123.34EE10
0.123
digit —> [0-9]
int —>digit+
float —> (digit*).(digit+)

Example (Floating-point Numbers)

- Floating-point numbers can be described as

$$
\begin{aligned}
digit &\to [\mathtt{0}\text{-}\mathtt{9}] \\
digits &\to digit^{+} \\
number &\to digits\ (.\ digits)?\ (\mathtt{E}\ [\mathtt{+-}]?\ digits)?
\end{aligned}
$$

Outline

Assignment

Assignment

- Read Section 3.3.
- Exercises 3.4.1-3