Course: Compiler Construction

Week 2: Compiler Design Phases – Lexical Analysis

Lecturer: Martha Gichuki

Learning outcomes Week 2: Compiler Design Phases – Lexical Analysis

At the end of the lecture, you will be able to:

i. Define a token, pattern and lexeme
ii. Describe how tokens are specified
iii. Explain the role of lexical analyzers.

Introduction to Compilation process

There are two parts in the compilation process

i. The **analysis part** – breaks up the source program into constant pieces and creates an intermediate representation of the source program **(Front-end)**.

ii. **Synthesis part** – constructs desired target program from the intermediate representation **(Back-end)**[1].

Phases of a Compiler

To ease the process of development and understanding, a compiler can be conceptually divided

[1] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 29

into the following phases[2]:

i. Lexical analysis
ii. Syntax analysis
iii. Semantic analysis
iv. Intermediate code generation
v. Target code generation
vi. Code optimization
vii. Symbol table management
viii. Error handling and recover

Lexical Analysis

This is the initial part of reading and analyzing the program text; the text is read and divided into tokens, each of which corresponds to a symbol in the programming language e.g. a variable name, number, keyword, etc. It is basically used to identify valid words in the input source program.
The word "***lexical***" in traditional sense, means "***pertaining to words***". In terms of programing languages, words are objects like variable names, numbers, keywords etc. such words are traditionally called tokens[3] .

Three roles of a Lexical Analyzer

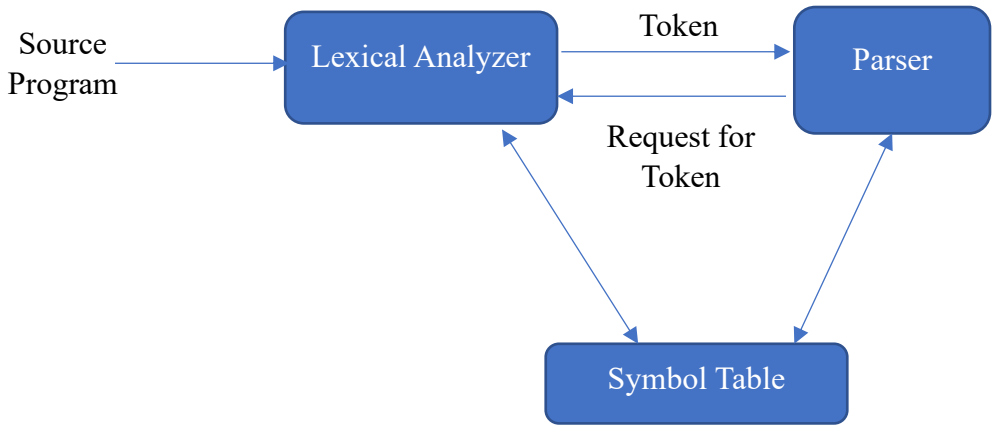
i. Takes as its input a string of individual words and divides them into tokens.

ii. Filters out layout-out characters (spaces, new lines etc.) and comments.

[2] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 30

[3] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 134

iii. Participates in creation and maintenance of a symbol table as shown below[4]:



Regular Expressions

Lexical analysis specifications are traditionally written using regular expressions. A regular expression is an algebraic notation used to describe sets of strings OR a notation used to specify patterns corresponding to a token.Once the set of regular expressions are ready, a finite automaton is constructed to determine whether or not a given word belongs to the language[5].

Definitions

A. **Token** - a lexical token is a sequence of characters that can be treated as a unit in the grammar of a programming language[6].

   Example: - a type token can consist of an identifier, number etc.

[4] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 134

[5] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 136

[6] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 136

Examples of non-tokens include[7]:

- ✓ comments,
- ✓ macros,
- ✓ pre-processor directives,
- ✓ tabs,
- ✓ new lines etc.

B. **Patterns -** this is a set of strings for which the same token is produced as output[8]. Regular expressions are an important notation for specifying patterns.
Example: - the pattern for Pascal identifier token id is:
id→letter(letter|digit) *

C. **Lexeme** - this is a sequence of characters in the source program that is matched by the pattern for a token[9].
Example: - the lexer will return the same token to the parser whenever it comes across the pattern that contains the following six lexemes: (=, <>, <, <=,>,>=)

Token Specifications

- ✓ The set of all variable names are sets of strings, where the individual letters are taken from a particular alphabet. Such a set of strings is called a **language**[10].
- ✓ Therefore, a language is ***a set of finite length sequences of elements*** drawn from a specified finite set **A** of symbols[11].

[7] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 136
[8] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 137
[9] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 136
[10] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 136
[11] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 136

- ✓ A language is analogous to a collection of words.
- ✓ In this case, the **set A** is called the alphabet of **L**, whose elements are called **words**.
- ✓ An alphabet might be {**a,b**}, and a string of that alphabet might be **ababba**.
- ✓ The empty word (**length zero string**) is allowed and is often denoted by $\boldsymbol{e}$, $\boldsymbol{\varepsilon}$ or $\boldsymbol{\Lambda}$[12].
- ✓ Given an alphabet, it is possible to describe a set of strings using a regular expression i.e. regular expressions of a language are created by combining its alphabet.[13]

For a language, **L**, with alphabet **set** $\sum$**,** the following rules define regular expressions:

1) $\boldsymbol{e}$ is a regular expression denoting the language $\{\boldsymbol{e}\}$ i.e. the set containing only the empty string[14].
2) If '**a**' is a special symbol in $\sum$ then '**a**' denotes a regular expression corresponding to the language $\{\boldsymbol{a}\}$ i.e. set containing only string '**a**'[15].
3) If $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are regular expressions corresponding to the languages $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ respectively then[16]:

    a) $\boldsymbol{r_1|r_2}$ is a regular expression corresponding to the language $\boldsymbol{L_1 \cup L_2}$ i.e. set containing all strings in: L_1 and L_2
    b) $\boldsymbol{r_1 r_2}$ is a regular expression corresponding to the language created by **concatenating** strings of L_2 to strings of L_1
    c) $\boldsymbol{r_1}$* is a regular expression corresponding to the language $\boldsymbol{L_1}$* i.e. the set containing zero or more occurrences of the strings belonging to $\boldsymbol{L_1}$

[12] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 147
[13] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 146
[14] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 146
[15] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 145
[16] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 145

d) (r_1) is a regular expression corresponding to the language L_1 itself[17]

Operator Order of Precedence

The **unary operator *** has the highest precedence followed by **concatenation** while **alteration** '|' has the lowest precedence[18]

Examples of Regular Expressions

Consider the following regular expressions for a simple alphabet consisting of only two letters: **Σ = {0,1}**

i. **(0|1) *** will be all the binary strings including the empty string

ii. **(0|1) (0|1) *** will be all non-empty binary strings[19].

iii. **(0|1) *0** denotes all strings of length of at least two, starting and ending with zeros[20].

iv. **(0|1) *0(0|1) (0|1)** denotes all the binary strings with at least three characters, where the third last character is always zero.

v. **0*10*10*10*** denotes all binary strings possessing exactly three ones[21]

[17] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 145
[18] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 145
[19] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 147
[20] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 147
[21] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 147

- ✓ Most programming languages define variables as **letter(letter|digit|'_') *** i.e. a variable name is a sequence of letters, digits and the '_' but the first character must be a letter[22].

- ✓ The set of signed integers may be specified as:

  (+|e |) digit(digit)*

  or

  (+| |e) [0-9] ([0-9]) *

Exercise 1

a) Write down the regular expression for the following:

Binary strings such as '0' followed by '1'

Solution: 01*

String examples: 01, 011, 0111 …

b) To check an IP address for correct syntax, four groups of 1 - 3 digits separated by periods

Solution: ***Octet:***	25[0-5]	**:**2[0-4][0-9]	**:**[01][0-9][0-9]	**:**[0-9]
Range:	250-255	200-249	0-199	0-9

Example of an IP address 255:233:89:9 - full colon to separate the octet ranges

a) Any decimal number that is a multiple of 5

Solution: Last digit is zero or five. Example 05, 10, 30,25 …

Exercise 2

Describe the following regular expressions

a) $a^*(a|b)$

Solution: Zero or more occurrences of **a** then a sequence of **a** alternating with **b**. String ends with either **a** or **b**. Not a must we start with **a**. String examples: a, b, aa, ab, aaa, aab …

b) $(a|b)^*abb$

Solution: Zero or more occurrences of **a** or **b at the start,** then string ends with **abb**. String examples: abb, aabb, abababb, aaabb, aababb …

[22] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 146

Content Covered in Week 2: Compiler Design Phases – Lexical Analysis

At the end of the lecture, we were able to:

i. Define a token, pattern and lexeme

ii. Describe how tokens are specified

iii. Explain the role of lexical analyzers.

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007))

2. Compiler Construction: Principles and practices; Kenneth C. Louden; Cengage Learning; 1st edition, ISBN-10 : 0534939724 (1997)

3. Basics of Compiler Design: Torben Mogensen; DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK (2007)

4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; Morgan Kaufmann Publishers ISBN: 978-0-12-088478-0 (2003)

5. Compiler Design: Santanu Chattopadhyay; PHI Learning Publishers, ISBN 812032725X, 9788120327252 (2005)