Software Re-Engineering

Lecture: 11

Dr. Imran Ali
School of Computing- Software Engineering
FAST-National University of Computer & Emerging Sciences, Karachi

Sequence [Todays Agenda]

Content of Lecture

- Lexical Analysis

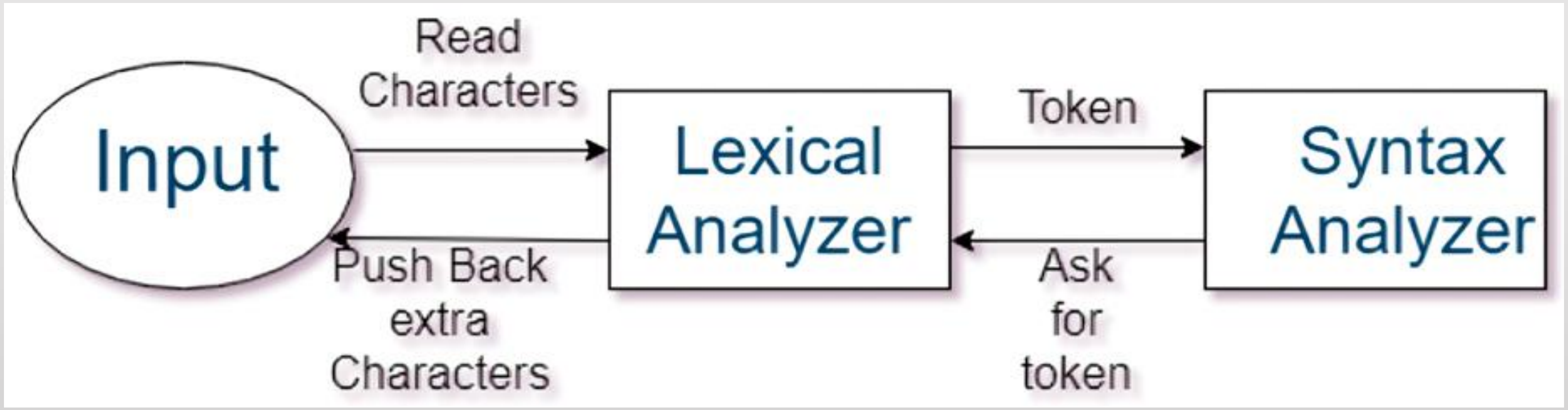
Lexical Analysis

- It is the process of decomposing the sequence of characters in the source code into its constituent lexical units (Tokens).
- It is the first phase of the compiler also known as a scanner or lexer or tokenizer.
  - Lexical Analysis can be implemented with the **Deterministic finite Automata**.
  - The output is a sequence of tokens that is sent to the parser for **syntax analysis**.
- Lexical analysis is an important first step in Natural Language Processing. In programming, this process involves a tool called a lexical analyzer, or lexer, which reads the source code one character at a time.

Deterministic Finite Automata

- A Deterministic Finite Automata (DFA) is a mathematical model used to recognize patterns in strings of characters.
- Key components of DFA are:
  - Finite set of states (Q),
  - Alphabet Σ,
  - Transition function δ,
  - Initial state q_0, and
  - Accepting states F.
- For each state and input symbol, a DFA transitions to exactly one next state, making it deterministic.

Lexical Analysis



Token

- A lexical token is a sequence of characters that can be treated as a unit in the grammar of the programming languages.
- **Examples of tokens:**
  - Keywords: while, for, if, else etc.
  - Identifiers: Variable name, function name, etc.
  - Operators: '+', '++', '-' etc.
  - Separators: ',' ';' etc.
- **Examples of non-tokens:**
  - Comments, preprocessor directive, macros, blanks, tabs, newline, etc.

Lexeme

- A lexeme is a sequence of characters of a program (source code) that is grouped together as a single unit.
- A lexeme is a basic unit of meaning in a language.
- It represents the core idea of a word.
- For example, the words “write,” “wrote,” “writing,” and “written” all relate to the same lexeme: “write.”
- In a simple expression like “5 + 6,” the individual elements “5,” “+,” and “6” are separate lexemes.
- (,) , { , } are lexemes of type punctuation (token).

Lexical Analyzer

- **How Lexical Analyzer Works?**
- **Input preprocessing:**
- This stage involves cleaning up the input text and preparing it for lexical analysis.
- This may include removing comments, whitespace, and other nonessential characters from the input text.
- **Tokenization:**
- This is the process of breaking the input text into a sequence of tokens.
- This is usually done by matching the characters in the input text against a set of patterns or regular expressions that define the different types of tokens.

- **Token classification:**
- In this stage, the lexer determines the type of each token.
- For example: In a programming language, the lexer might classify keywords, identifiers, operators, and punctuation symbols as separate token types.
- **Token validation:**
- In this stage, the lexer checks that each token is valid according to the rules of the programming language.
- For example: It might check that a variable name is a valid identifier, or that an operator has the correct syntax.

- **Output generation :**
- In this final stage, the lexer generates the output of the lexical analysis process, which is typically a list of tokens.
- This list of tokens can then be passed to the next stage of compilation or interpretation.

- *The lexical analyzer identifies the error with the help of the automation machine and the grammar of the given language on which it is based like C, C++, and gives row number and column number of the error.*

Steps involved in Lexical Analyzer

- Lexical analysis is a process that breaks down an input text into small parts called tokens or lexemes.
- This helps computers to understand the text for further analysis.
- Although the exact steps can vary depending on the organizational requirement and complexity of the text but most of the processes follow these basic steps:
- Step 1. Identify Tokens
- Step 2. Assign Strings to Tokens
- Step 3. Return Lexeme or Value

- Step 1. Identify Token:
- The first step involved in lexical analysis is to identify individual symbols in the input.
- These symbols include letters, numbers, operators, and special characters.
- Each symbol or group of symbols is given a token type, like “number” or “operator.”

- <u>Step 2. Assign Strings to Tokens:</u>
- The lexer (a tool that performs lexical analysis) is set up to recognize and group-specific inputs.
- For example, if the input is “apple123,” the lexer may recognize “apple” as a word token and “123” as a number token.
- Similarly, keywords like “if” or “while” are categorized as specific token types.

- Step 3. Return Lexeme or Value:
- The lexer breaks the input into the smallest meaningful parts called lexemes.
- It then returns these lexemes along with their token type.
- This process helps the next stage of analysis which is to understand what each token represents in the text.

Different Types of Lexical Analysis

- When choosing a method for lexical analysis, there are two main approaches:

1. "Loop and Switch" and
2. "Regular Expressions with Finite Automata."

- Both methods help users to analyze input text by breaking it into smaller parts called tokens, making it easier for computers to process.

Loop and Switch

- The **loop** works like reading a book, one character at a time until it reaches the end of the code. It goes through the code without missing any character or symbol, making sure each part is captured.

- The **switch statement** acts as a quick decision-maker. After the loop reads a character or a group of characters, the switch statement decides what type of token it is, such as a keyword, number, or operator.
- This is similar to organizing items into separate boxes based on their type, making the code easier to understand.
- For example, if the loop reads “cat”, the switch statement quickly decides it’s a string or keyword token.

Regular Expressions with Finite Automata

- **Regular expressions** are rules that describe patterns in text.
- They help define how different tokens should look, such as an email or a phone number format.
- The lexer uses these rules to identify tokens by matching text against these patterns.

- **Finite automata** are like small machines that follow instructions step-by-step.
- They take the rules from regular expressions and apply them to the code.
- If a part of the code matches a rule, it’s identified as a token. This makes the process of breaking down code more efficient and accurate.

Advantages of Lexical Analysis

- **Cleans Up Data:** Lexical analysis removes unnecessary elements like extra spaces or comments from the text, making the text cleaner and easier to work with.
- **Simplifies Further Analysis:** It breaks the text into smaller units called tokens and filters out irrelevant data, making it easier to perform other analyses like syntax checking.
- **Reduces Input Size:** It helps in compressing the input data by organizing it into tokens, this simplifies and speeds up the text processing.

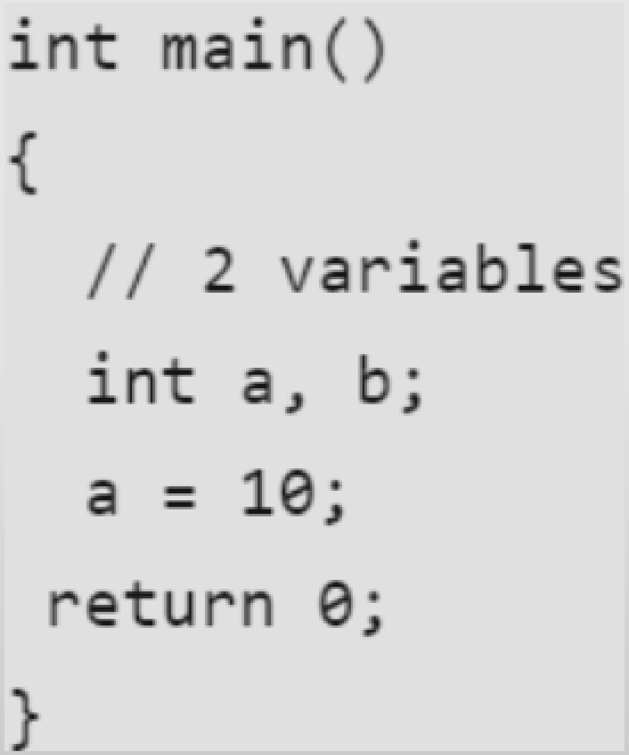
Disadvantages of Lexical Analysis

- **Limited Context:** Lexical analysis operates based on individual tokens and does not consider the overall context of the code. This can sometimes lead to ambiguity or misinterpretation of the code’s intended meaning especially in languages with complex syntax or semantics.
- **Overhead:** Although lexical analysis is necessary for the compilation or interpretation process, it adds an extra layer of overhead. Tokenizing the source code requires additional computational resources which can impact the overall performance of the compiler or interpreter.

- **Debugging Challenges:** Although lexical analysis Lexical errors detected during the analysis phase may not always provide clear indications of their origins in the original source code. Debugging such errors can be challenging especially if they result from subtle mistakes in the lexical analysis process.
- Subtle mistakes in lexical analysis, particularly regarding token identification, can lead to misinterpretations of source code. For example, the "longest-match rule" can sometimes incorrectly combine multiple tokens into one, or fail to recognize valid tokens.

Lexical Analysis Examples

- Example 1:

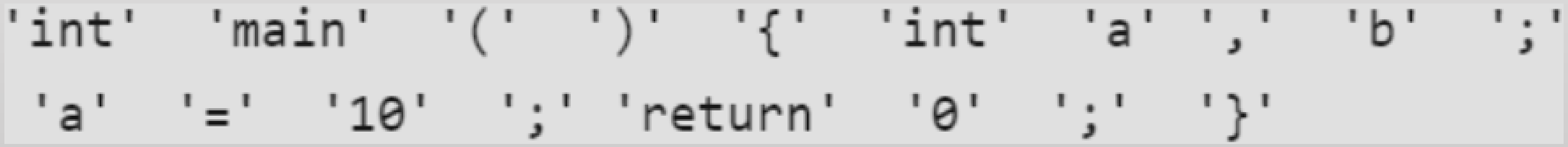


```
int main()
{
  // 2 variables
  int a, b;
  a = 10;
 return 0;
}
```

- **Valid Output Tokens ?**

Lexical Analysis Examples

- Valid Output Tokens:
- ‘ int ’ ‘main’ ‘(’ ‘)’ ‘{’ ‘int’ ‘a’ ‘,’ ‘b’ ‘;’ ‘a’ ‘=’ ‘10’ ‘return’ ‘0’ ‘;’ ‘}’



```
'int'  'main'  '('  ')'  '{'  'int'  'a' ','  'b'  ';'
 'a'  '='  '10'  ';' 'return'  '0'  ';'  '}'
```

Thank You!