Software Re-Engineering

Lecture: 11

Dr. Syed Muazzam Ali Shah

Department of Software Engineering

National University of Computer & Emerging Sciences

muazzam.ali@nu.edu.pk

Sequence [Todays Agenda]

Content of Lecture

Reverse Engineering – Techniques

- **Lexical Analysis**
- Syntactic Analysis
- Control Flow Analysis
- Data Flow Analysis
- Program Slicing
- Visualization
- Program metrics

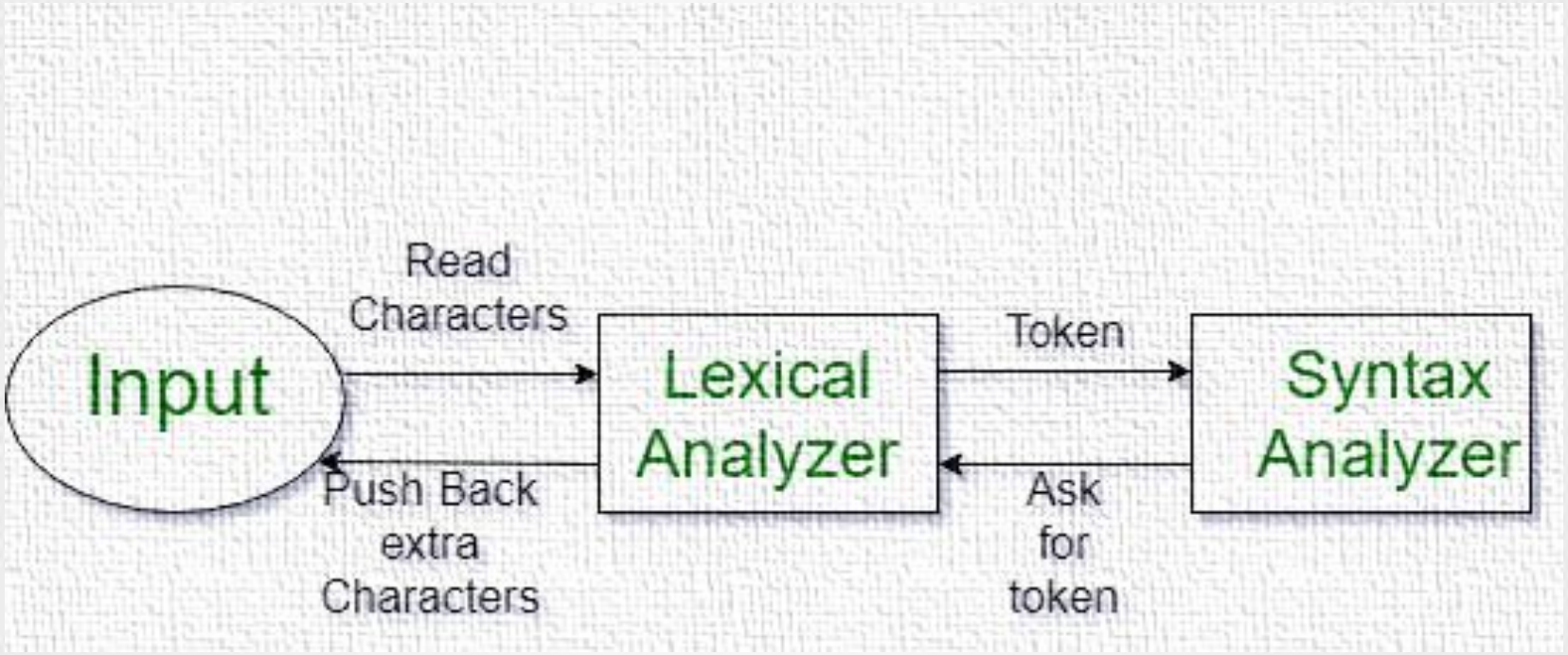
Reverse Engineering – Techniques

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.**

Reverse Engineering – Techniques

Lexical Analysis



Reverse Engineering – Techniques

Lexical Analysis

❖ What is a 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.

Reverse Engineering – Techniques

Lexical Analysis

❖ Lexeme?

- A lexeme is a sequence of characters of a program (source code) that is grouped together as a single unit.
- **Examples of lexems:**

```
main is lexeme of type identifier(token)
(,),{,} are lexemes of type punctuation(token)
```

Reverse Engineering – Techniques

Lexical Analysis

❖ 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 non-essential 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.

Reverse Engineering – Techniques

Lexical Analysis

❖ How Lexical Analyzer Works?

- **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.

Reverse Engineering – Techniques

Lexical Analysis

❖ How Lexical Analyzer Works?

- **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.

Reverse Engineering – Techniques

Lexical Analysis

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.

Reverse Engineering – Techniques

Lexical Analysis

- For example:

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

- Valid output Tokens:

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

Reverse Engineering – Techniques

Lexical Analysis

- For example:

Printf (“Welcome to FAST-NUCES”);

Reverse Engineering – Techniques

Lexical Analysis

- For example:

```
Printf ( "Welcome to FAST-NUCES" ) ;
1      2  3                       4  5
```

Reverse Engineering – Techniques

Lexical Analysis

- For example:

```
int main()
{
  int a = 10, b = 20;
  printf("sum is:%d",a+b);
  return 0;
}
```

- Valid output Tokens:

```
Answer: Total number of token: 27.
```

Reverse Engineering – Techniques

Lexical Analysis

❖ For example:

- **Exercise: Count number of tokens:**

  int max(int i);

- **Solution**
  - Lexical analyzer first read **int** and finds it to be valid and another token and finally ;

```
Answer:  Total number of tokens 7:
int, max, ( ,int, i, ), ;
```

Reverse Engineering – Techniques

Lexical Analysis

- We can represent in the form of lexemes and tokens as under

Lexemes	Tokens	Lexemes	Tokens
while	WHILE	a	IDENTIEFIER
(	LAPREN	=	ASSIGNMENT
a	IDENTIFIER	a	IDENTIFIER
>=	COMPARISON	–	ARITHMETIC
b	IDENTIFIER	2	INTEGER
)	RPAREN	;	SEMICOLON

Reverse Engineering – Techniques

Lexical Analysis

- Advantages

Simplifies Parsing:

- Breaking down the source code into tokens makes it easier for computers to understand and work with the code.
- This helps programs like compilers or interpreters to figure out what the code is supposed to do. It's like breaking down a big puzzle into smaller pieces, which makes it easier to put together and solve.

Error Detection:

- Lexical analysis will detect lexical errors such as misspelled keywords or undefined symbols early in the compilation process.
- This helps in improving the overall efficiency of the compiler or interpreter by identifying errors sooner rather than later.

Reverse Engineering – Techniques

Lexical Analysis

- Advantages
  - **Efficiency:**
    - Once the source code is converted into tokens, subsequent phases of compilation or interpretation can operate more efficiently.
    - Parsing and semantic analysis become faster and more streamlined when working with tokenized input.

Reverse Engineering – Techniques

Lexical Analysis

❖ Disadvantages

- **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.

Reverse Engineering – Techniques

Lexical Analysis

- Disadvantages
  - **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.


Thank You!