## CS5800 – theory of foundations

## Western Michigan University

Instructor: Dr. Elise de Doncker

TA: Edwin Jose

Team: Sri Harsha Godavarthi, Venu Modugula

Sathish Kumar madiga Mallayolla

**Lexical Analysis Phase**

Objective: The lexical analysis phase, often referred to as the scanner, transforms a sequence of characters into a sequence of tokens. Tokens are categorized according to their role in the language's grammar, such as keywords, identifiers, operators, separators, and others. This phase also handles the detection of illegal characters.

**Test Cases:**

**Keywords Identification**

Input: Class, while, for, constructor

Output: keywords

Explanation: Reserved words like Class, while, for, and constructor are predefined by the language's grammar and should be recognized as keywords, indicating they have a special meaning in the language.

**Identifiers**

**Input:** Variables used by the programmer (e.g., sum, list)

**Output:** identifiers

Explanation: Identifiers are names given to elements like variables and functions. This test case ensures that the scanner can distinguish user-defined names from other token classes.

Arithmetic Operators

Input: Plus (+), Minus (-), Multiplication (\*), Division (/)

Output: Arithmetic Operator String (e.g., PLUS for +, DIV for /)

Explanation: Arithmetic operators should be recognized and labeled according to their function in mathematical expressions.

Logical Operators

Input: logical\_AND (&&), logical\_OR (||) etc..

Output: LOGICAL\_<operator\_name>

Explanation: Logical operators used in conditional statements and boolean expressions must be correctly identified, such as LOGICAL\_AND for &&.

Separators

Input: ;, {}, (), []

Output: SEPARATOR

Explanation: Separators like semicolons, brackets, and parentheses play a critical role in defining the structure of statements and blocks in the program.

Illegal Characters

Input: cl<invalidsymbol\_in\_language>ass

Output: Illegal character: <character symbol>

Explanation: The presence of characters not recognized by the language's grammar should trigger an error, identifying the specific illegal character.

**Parser Phase**

Objective: The parser phase analyzes the sequence of tokens from the lexical phase according to the grammar of the language, constructing a parse tree or abstract syntax tree (AST). This phase checks the syntactic structure of the program and handles syntax errors.

Test Cases:

Valid Syntax

Input: Keyword <identifier> LPR (e.g., int identifier = number;)

Output: AST representing left-most derivation, displaying each level.

Explanation: This case tests the parser's ability to recognize and correctly parse declarations and assignments, building an appropriate AST.

Invalid Syntax

Input: identifier expression

Output: invalid syntax

Explanation: When the input deviates from the expected grammatical structure, the parser should report an error. This case checks the parser's error-reporting mechanism for unexpected or misplaced tokens.

Syntax Error Handling

Missing Delimiters

Case1: A missing semicolon at the end of a statement.

Explanation: The parser should detect and report missing delimiters that are critical for the correct separation of statements.

Incorrect Use of Keywords

Case2: Using class keyword instead of void.

Explanation: The misuse of reserved keywords in incorrect contexts should be flagged as a syntax error.

Unbalanced Parentheses or Braces

Explanation: The parser must ensure that every opening parenthesis or brace has a corresponding closing counterpart. Imbalance in these symbols often indicates a syntactic error.

Additional Test Cases

Reading Input from File

Input: Reading input.txt containing Java code in string format.

Output: Java code as a string.

Explanation: This case tests the system's ability to read and correctly interpret the contents of a file as input for further processing.

Evaluating Expressions

Input: x+y/z+(k+z)

Output: Output based on operator precedence given in the program.

Explanation: This test ensures that the parser correctly applies operator precedence rules when constructing the AST, which is crucial for accurate expression evaluation