DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY

UNITY · FAITH · DISCIPLINE

Name : Umer Khan
Roll no: E-20/F-BSCS-20

Name: Sabeer Junaid
Roll no: E-20/F-BSCS-26

Course: Compiler Design
Course Instructor: Dr. Asma Sanam Larik

Assignment

```
import java.io;
import java.util;
import java.util.regex;

public class Lexer {
    // Define regular expressions for tokens
    private static final Set<String> keywords = new HashSet<>(Arrays.asList("if", "else", "while",
"for",
            "int", "float", "char", "return"));
    private static final String identifierRegex = "[a-zA-Z_]\\w*";
    private static final String integerRegex = "\\d+";
    private static final String stringRegex = "\"([^\"\\\\]|\\\\.)*\"";
    private static final String charRegex = "'([^'\\\\]|\\\\.)'";
    private static final String commentRegex = "/\\.?\\*/";
    private static final String symbolRegex = "[\\[\\]{}(),;=+\\-*/<>!]";

    // Function to tokenize a line of code
    private static List<Token> tokenizeLine(String line, int lineNumber) {
        List<Token> tokens = new ArrayList<>();
        int position = 0;
        while (position < line.length()) {
            Matcher matcher = null;
            for (String regex : Arrays.asList(commentRegex, stringRegex, charRegex,
identifierRegex,
                    integerRegex, symbolRegex)) {
                Pattern pattern = Pattern.compile(regex);
                matcher = pattern.matcher(line.substring(position));
                if (matcher.find()) {
                    String lexeme = matcher.group();
                    String tokenType;
                    switch (regex) {
                        case identifierRegex:
                            tokenType = keywords.contains(lexeme) ? "KEYWORD" : "IDENTIFIER";
                            break;
                        case integerRegex:
                            tokenType = "INTEGER";
                            break;
                        case stringRegex:
                            tokenType = "STRING";
                            break;
                        case charRegex:
                            tokenType = "CHAR";
                            break;
                case commentRegex:
                    tokenType = "COMMENT";
                    break;
                default:
                    tokenType = "SYMBOL";
            }
            tokens.add(new Token(tokenType, lexeme, lineNumber));
            position += lexeme.length();
            break;
        }
    }
    if (matcher == null || !matcher.find()) {
        position++; // Skip invalid character
    }
  }
  return tokens;
}

// Function to handle specific language features and error handling
private static List<Token> analyzeCode(String fileName) throws IOException {
    List<Token> tokens = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        String line;
        int lineNumber = 0;
        while ((line = reader.readLine()) != null) {
            lineNumber++;
            // Remove comments
            line = line.replaceAll(commentRegex, "");
            // Tokenize line
            tokens.addAll(tokenizeLine(line, lineNumber));
        }
    }
    return tokens;
}

// Main function
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the name of the source code file: ");
    String fileName = scanner.nextLine();
    try {
        List<Token> tokens = analyzeCode(fileName);
        try (PrintWriter writer = new PrintWriter(new FileWriter("output.txt"))) {
            writer.println("Token\tLexeme\tLine No");
```

```
                for (Token token : tokens) {
                    writer.println(token.getType() + "\t" + token.getLexeme() + "\t" +
token.getLineNumber());
                }
            }
            System.out.println("Lexical analysis completed. Results saved in output.txt.");
        } catch (FileNotFoundException e) {
            System.out.println("File not found.");
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }

    static class Token {
        private String type;
        private String lexeme;
        private int lineNumber;

        public Token(String type, String lexeme, int lineNumber) {
            this.type = type;
            this.lexeme = lexeme;
            this.lineNumber = lineNumber;
        }

        public String getType() {
            return type;
        }

        public String getLexeme() {
            return lexeme;
        }

        public int getLineNumber() {
            return lineNumber;
        }
    }
}
```