Kingdom of Saudi Arabia
Ministry of Education
Umm Al-Qura University
Collage of Engineering & Computing at Al-Qunfudah
Computing Department
Computer Science Program

Course Project for Compiler Construction (CS2341)

Group Members Contribution to the Project.			
Name Student	Id	Section	Task
Ahad Suleiman Al-marhabi	444002198	1	Lexer and source code
Lujain hassan Al-Kinani	444007316	1	Parser and Introduction
Raghad Hassan AL-Masari	444001447	1	Lexer and source code
Yara Ahmad Alzailai	444010350	1	Parser and Introduction

Supervised by Dr. Essa Muharish

Introduction:

A compiler is a program that converts source code, written in an easy-to-understand programming language, into a low-level form that a computer can understand. The compiler does this by preserving the original meaning and intent of the source code.

The front-end compiler, an important part of the compiler, converts the source code into a format that can be processed by the computer. It is necessary to convert human-readable code into a form that a computer can execute.

In our project, the source code is code to calculate the area of a rectangle that contains the following variables:

(len): represents the length of the rectangle (5)

(wid): represents the width of the rectangle (4)

Through the law of calculating the rectangle, length * width, the result is improved and printed.

Project summary:

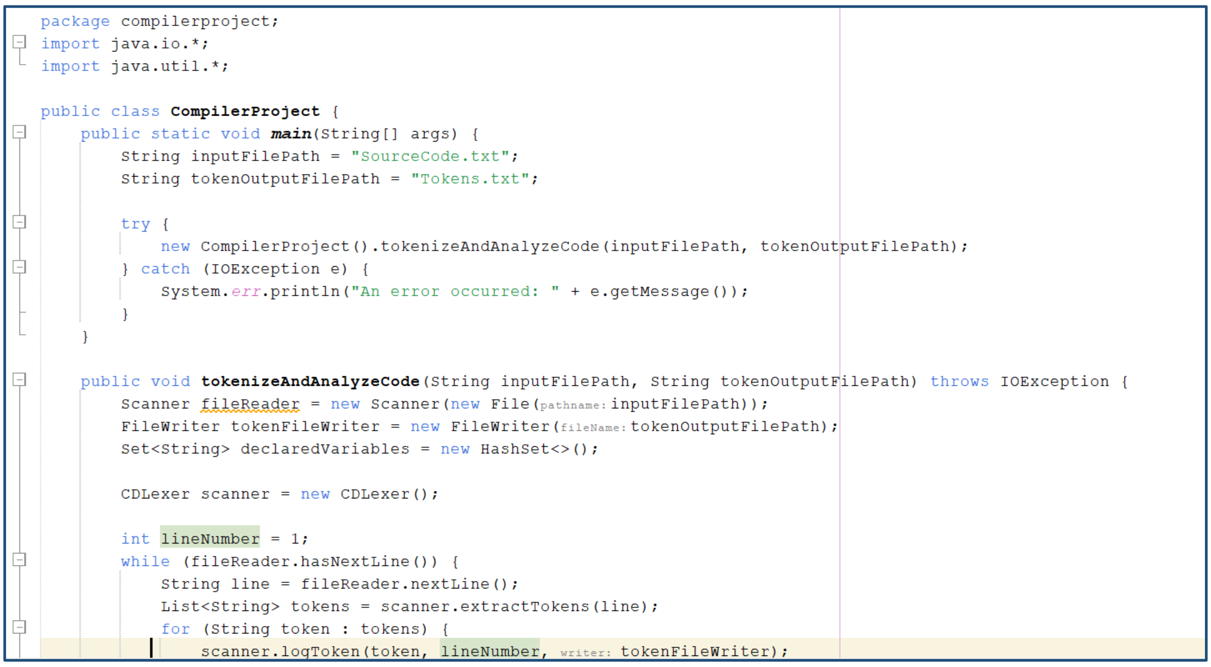
can use any programming language you choose to implement a front-end compiler that should include lexical analysis, parsing, building abstract syntax, type-checking, and a few static checks.

To implement a compiler front end:

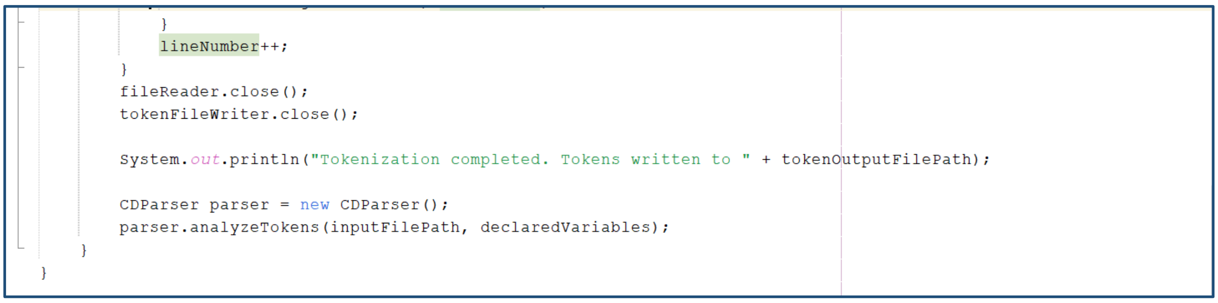
1. Define suitable data types/classes for representing abstract syntax.
2. Implement a lexer and parser that builds abstract syntax from strings.
3. Implement a type checker that checks that programs are type-correct.
4. Implement a main program that calls lexer, parser and type checker, and reports errors.

Code of Compiler Project:

1- Main Class:

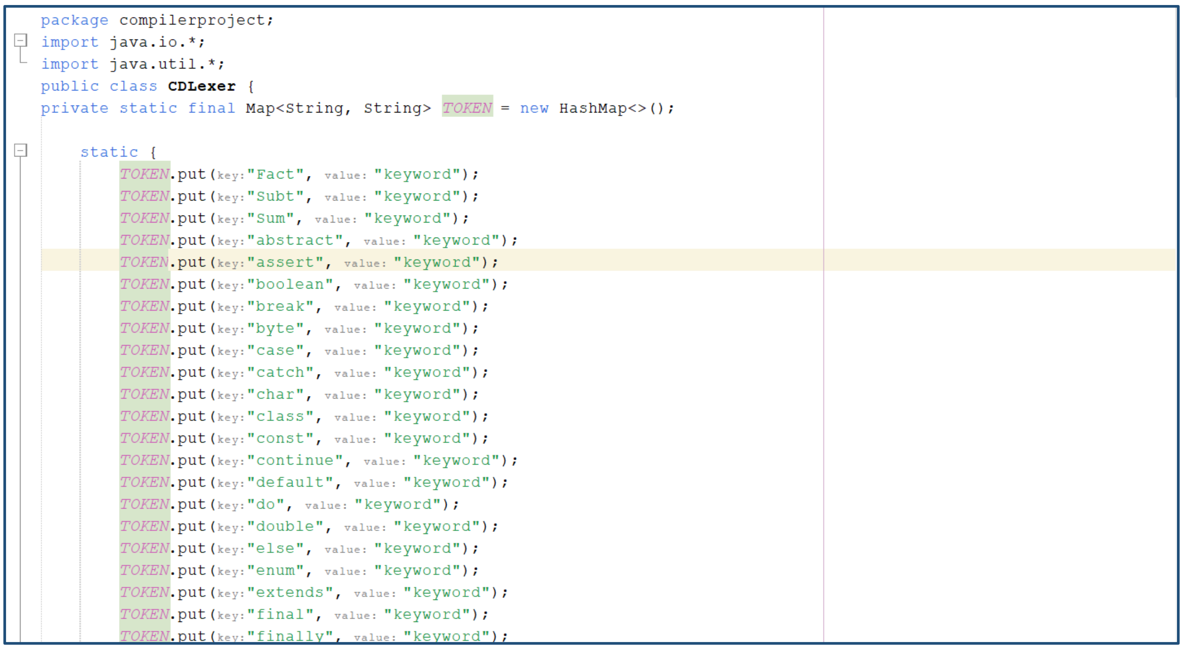


```
package compilerproject;
import java.io.*;
import java.util.*;

public class CompilerProject {
    public static void main(String[] args) {
        String inputFilePath = "SourceCode.txt";
        String tokenOutputFilePath = "Tokens.txt";

        try {
            new CompilerProject().tokenizeAndAnalyzeCode(inputFilePath, tokenOutputFilePath);
        } catch (IOException e) {
            System.err.println("An error occurred: " + e.getMessage());
        }
    }

    public void tokenizeAndAnalyzeCode(String inputFilePath, String tokenOutputFilePath) throws IOException {
        Scanner fileReader = new Scanner(new File(pathname: inputFilePath));
        FileWriter tokenFileWriter = new FileWriter(fileName: tokenOutputFilePath);
        Set<String> declaredVariables = new HashSet<>();

        CDLexer scanner = new CDLexer();

        int lineNumber = 1;
        while (fileReader.hasNextLine()) {
            String line = fileReader.nextLine();
            List<String> tokens = scanner.extractTokens(line);
            for (String token : tokens) {
                scanner.logToken(token, lineNumber, writer: tokenFileWriter);
```

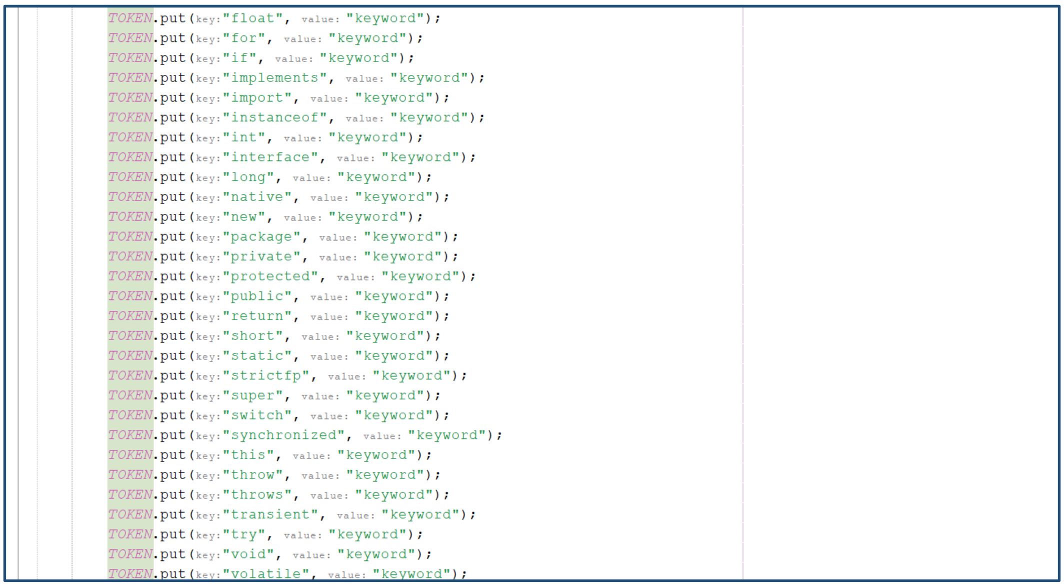


```
            }
            lineNumber++;
        }
        fileReader.close();
        tokenFileWriter.close();

        System.out.println("Tokenization completed. Tokens written to " + tokenOutputFilePath);

        CDParser parser = new CDParser();
        parser.analyzeTokens(inputFilePath, declaredVariables);
    }
}
```

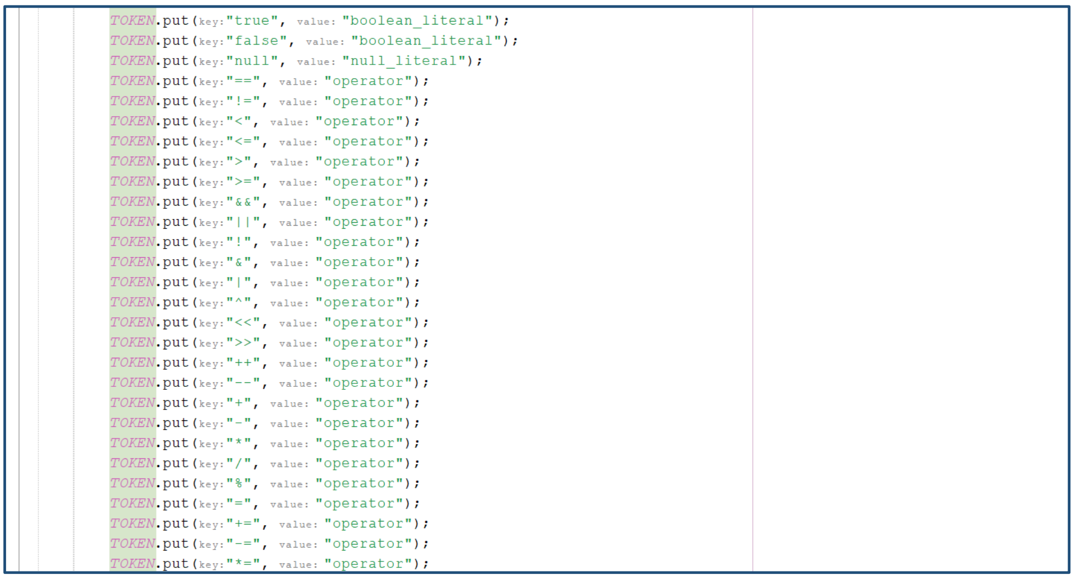
2- Code Lexer:



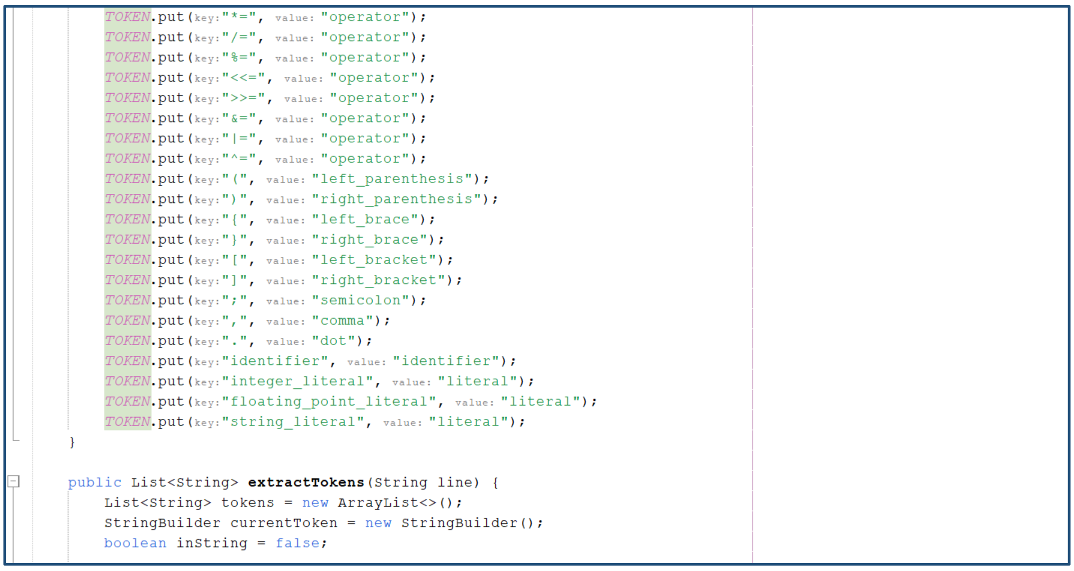
```
package compilerproject;
import java.io.*;
import java.util.*;
public class CDLexer {
private static final Map<String, String> TOKEN = new HashMap<>();

    static {
        TOKEN.put(key:"Fact", value: "keyword");
        TOKEN.put(key:"Subt", value: "keyword");
        TOKEN.put(key:"Sum", value: "keyword");
        TOKEN.put(key:"abstract", value: "keyword");
        TOKEN.put(key:"assert", value: "keyword");
        TOKEN.put(key:"boolean", value: "keyword");
        TOKEN.put(key:"break", value: "keyword");
        TOKEN.put(key:"byte", value: "keyword");
        TOKEN.put(key:"case", value: "keyword");
        TOKEN.put(key:"catch", value: "keyword");
        TOKEN.put(key:"char", value: "keyword");
        TOKEN.put(key:"class", value: "keyword");
        TOKEN.put(key:"const", value: "keyword");
        TOKEN.put(key:"continue", value: "keyword");
        TOKEN.put(key:"default", value: "keyword");
        TOKEN.put(key:"do", value: "keyword");
        TOKEN.put(key:"double", value: "keyword");
        TOKEN.put(key:"else", value: "keyword");
        TOKEN.put(key:"enum", value: "keyword");
        TOKEN.put(key:"extends", value: "keyword");
        TOKEN.put(key:"final", value: "keyword");
        TOKEN.put(key:"finally", value: "keyword");
```



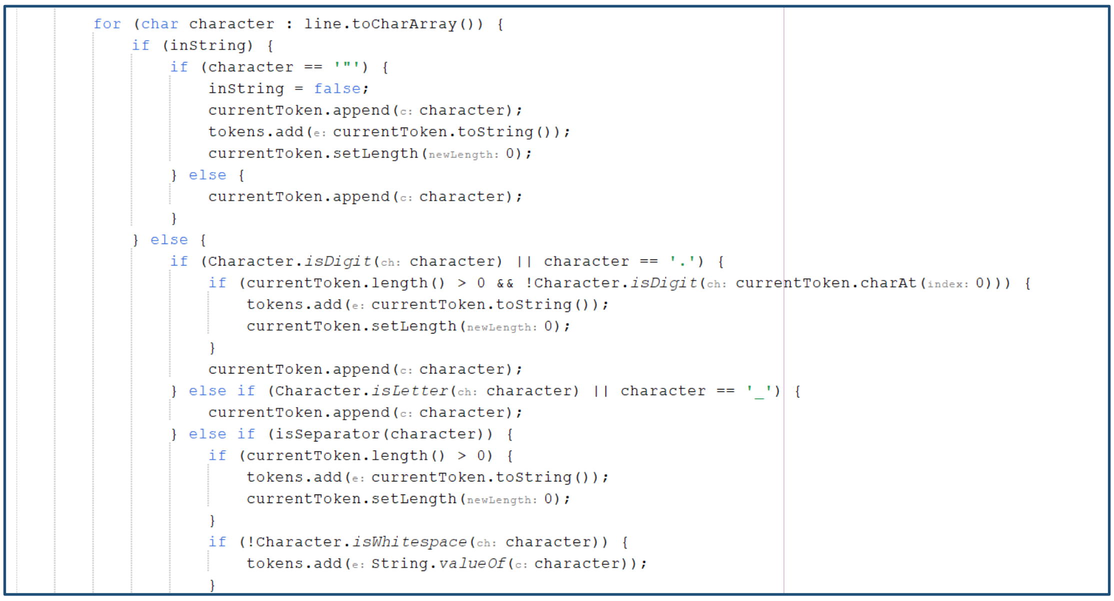
```java
        TOKEN.put(key:"float", value: "keyword");
        TOKEN.put(key:"for", value: "keyword");
        TOKEN.put(key:"if", value: "keyword");
        TOKEN.put(key:"implements", value: "keyword");
        TOKEN.put(key:"import", value: "keyword");
        TOKEN.put(key:"instanceof", value: "keyword");
        TOKEN.put(key:"int", value: "keyword");
        TOKEN.put(key:"interface", value: "keyword");
        TOKEN.put(key:"long", value: "keyword");
        TOKEN.put(key:"native", value: "keyword");
        TOKEN.put(key:"new", value: "keyword");
        TOKEN.put(key:"package", value: "keyword");
        TOKEN.put(key:"private", value: "keyword");
        TOKEN.put(key:"protected", value: "keyword");
        TOKEN.put(key:"public", value: "keyword");
        TOKEN.put(key:"return", value: "keyword");
        TOKEN.put(key:"short", value: "keyword");
        TOKEN.put(key:"static", value: "keyword");
        TOKEN.put(key:"strictfp", value: "keyword");
        TOKEN.put(key:"super", value: "keyword");
        TOKEN.put(key:"switch", value: "keyword");
        TOKEN.put(key:"synchronized", value: "keyword");
        TOKEN.put(key:"this", value: "keyword");
        TOKEN.put(key:"throw", value: "keyword");
        TOKEN.put(key:"throws", value: "keyword");
        TOKEN.put(key:"transient", value: "keyword");
        TOKEN.put(key:"try", value: "keyword");
        TOKEN.put(key:"void", value: "keyword");
        TOKEN.put(key:"volatile", value: "keyword");
```



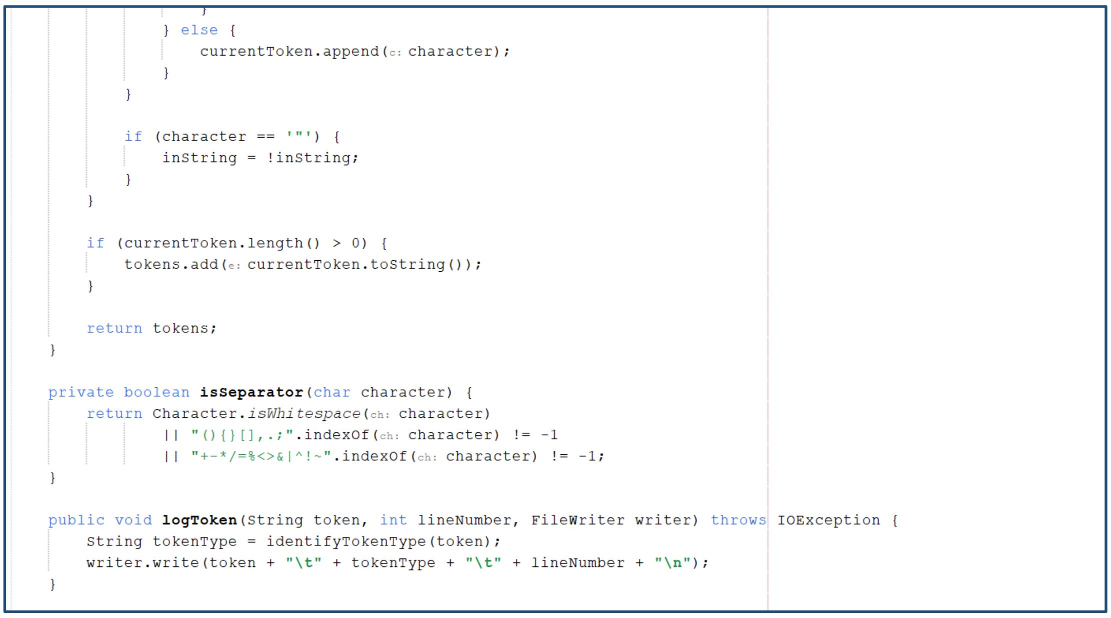
```java
        TOKEN.put(key:"true", value: "boolean_literal");
        TOKEN.put(key:"false", value: "boolean_literal");
        TOKEN.put(key:"null", value: "null_literal");
        TOKEN.put(key:"==", value: "operator");
        TOKEN.put(key:"!=", value: "operator");
        TOKEN.put(key:"<", value: "operator");
        TOKEN.put(key:"<=", value: "operator");
        TOKEN.put(key:">", value: "operator");
        TOKEN.put(key:">=", value: "operator");
        TOKEN.put(key:"&&", value: "operator");
        TOKEN.put(key:"||", value: "operator");
        TOKEN.put(key:"!", value: "operator");
        TOKEN.put(key:"&", value: "operator");
        TOKEN.put(key:"|", value: "operator");
        TOKEN.put(key:"^", value: "operator");
        TOKEN.put(key:"<<", value: "operator");
        TOKEN.put(key:">>", value: "operator");
        TOKEN.put(key:"++", value: "operator");
        TOKEN.put(key:"--", value: "operator");
        TOKEN.put(key:"+", value: "operator");
        TOKEN.put(key:"-", value: "operator");
        TOKEN.put(key:"*", value: "operator");
        TOKEN.put(key:"/", value: "operator");
        TOKEN.put(key:"%", value: "operator");
        TOKEN.put(key:"=", value: "operator");
        TOKEN.put(key:"+=", value: "operator");
        TOKEN.put(key:"-=", value: "operator");
        TOKEN.put(key:"*=", value: "operator");
```



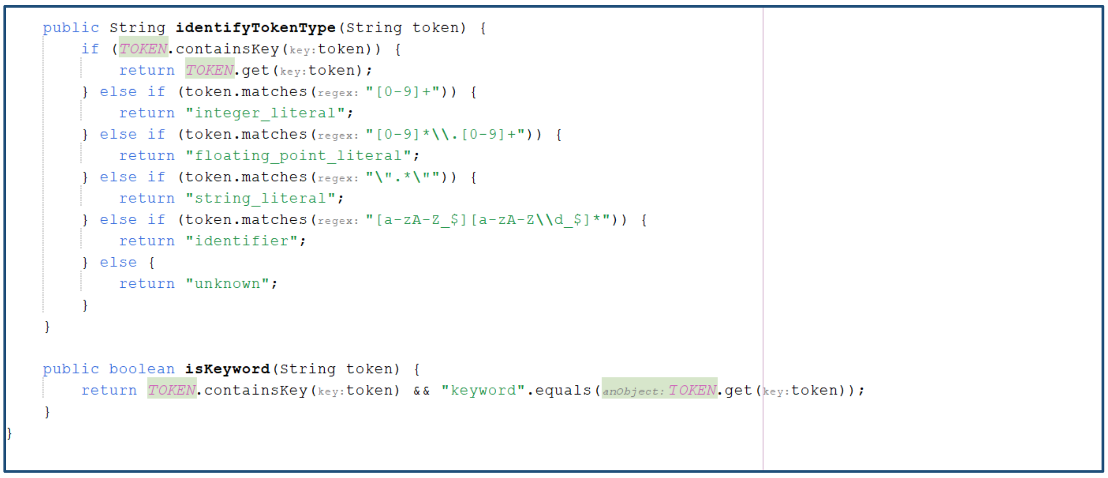
```java
        TOKEN.put(key:"*=", value: "operator");
        TOKEN.put(key:"/=", value: "operator");
        TOKEN.put(key:"%=", value: "operator");
        TOKEN.put(key:"<<=", value: "operator");
        TOKEN.put(key:">>=", value: "operator");
        TOKEN.put(key:"&=", value: "operator");
        TOKEN.put(key:"|=", value: "operator");
        TOKEN.put(key:"^=", value: "operator");
        TOKEN.put(key:"(", value: "left_parenthesis");
        TOKEN.put(key:")", value: "right_parenthesis");
        TOKEN.put(key:"{", value: "left_brace");
        TOKEN.put(key:"}", value: "right_brace");
        TOKEN.put(key:"[", value: "left_bracket");
        TOKEN.put(key:"]", value: "right_bracket");
        TOKEN.put(key:";", value: "semicolon");
        TOKEN.put(key:",", value: "comma");
        TOKEN.put(key:".", value: "dot");
        TOKEN.put(key:"identifier", value: "identifier");
        TOKEN.put(key:"integer_literal", value: "literal");
        TOKEN.put(key:"floating_point_literal", value: "literal");
        TOKEN.put(key:"string_literal", value: "literal");
    }

    public List<String> extractTokens(String line) {
        List<String> tokens = new ArrayList<>();
        StringBuilder currentToken = new StringBuilder();
        boolean inString = false;
```



```
        for (char character : line.toCharArray()) {
            if (inString) {
                if (character == '"') {
                    inString = false;
                    currentToken.append(c: character);
                    tokens.add(e: currentToken.toString());
                    currentToken.setLength(newLength: 0);
                } else {
                    currentToken.append(c: character);
                }
            } else {
                if (Character.isDigit(ch: character) || character == '.') {
                    if (currentToken.length() > 0 && !Character.isDigit(ch: currentToken.charAt(index: 0))) {
                        tokens.add(e: currentToken.toString());
                        currentToken.setLength(newLength: 0);
                    }
                    currentToken.append(c: character);
                } else if (Character.isLetter(ch: character) || character == '_') {
                    currentToken.append(c: character);
                } else if (isSeparator(character)) {
                    if (currentToken.length() > 0) {
                        tokens.add(e: currentToken.toString());
                        currentToken.setLength(newLength: 0);
                    }
                    if (!Character.isWhitespace(ch: character)) {
                        tokens.add(e: String.valueOf(c: character));
                    }
```

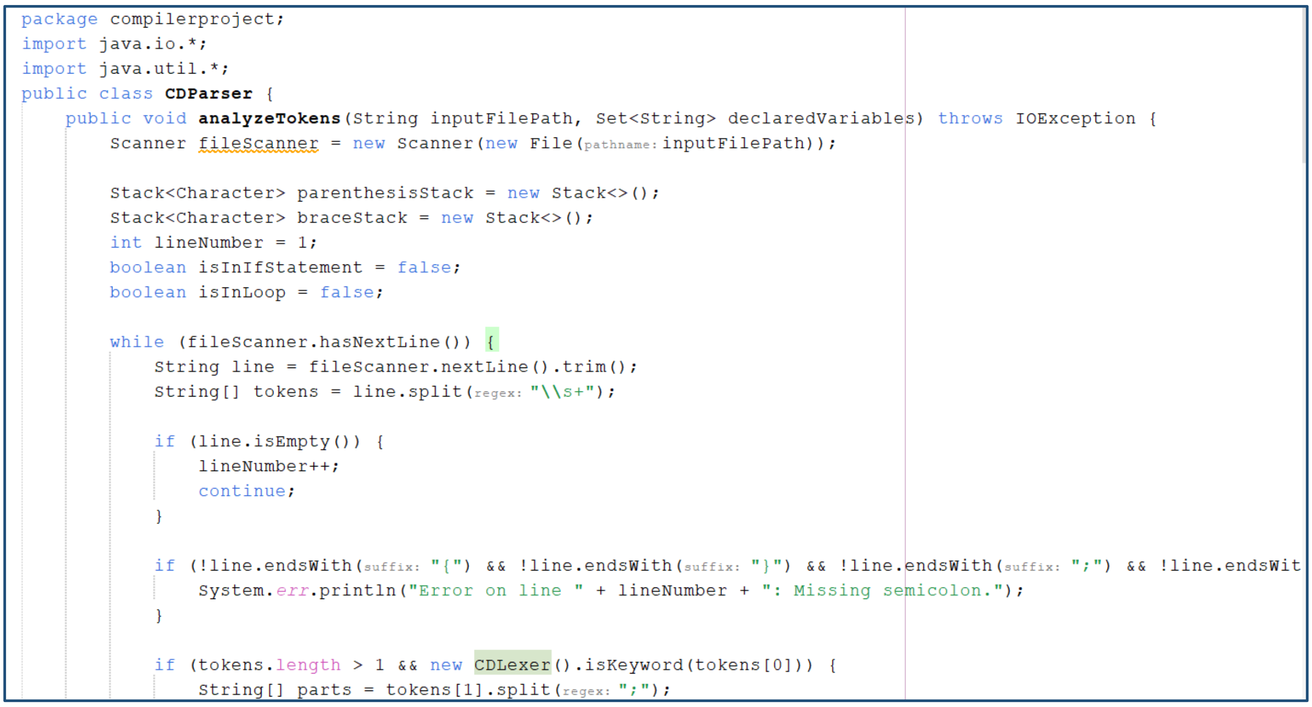


```
                } else {
                    currentToken.append(c: character);
                }
            }

            if (character == '"') {
                inString = !inString;
            }
        }

        if (currentToken.length() > 0) {
            tokens.add(e: currentToken.toString());
        }

        return tokens;
    }

    private boolean isSeparator(char character) {
        return Character.isWhitespace(ch: character)
                || "(){}[],.;".indexOf(ch: character) != -1
                || "+-*/=%<>&|^!~".indexOf(ch: character) != -1;
    }

    public void logToken(String token, int lineNumber, FileWriter writer) throws IOException {
        String tokenType = identifyTokenType(token);
        writer.write(token + "\t" + tokenType + "\t" + lineNumber + "\n");
    }
```

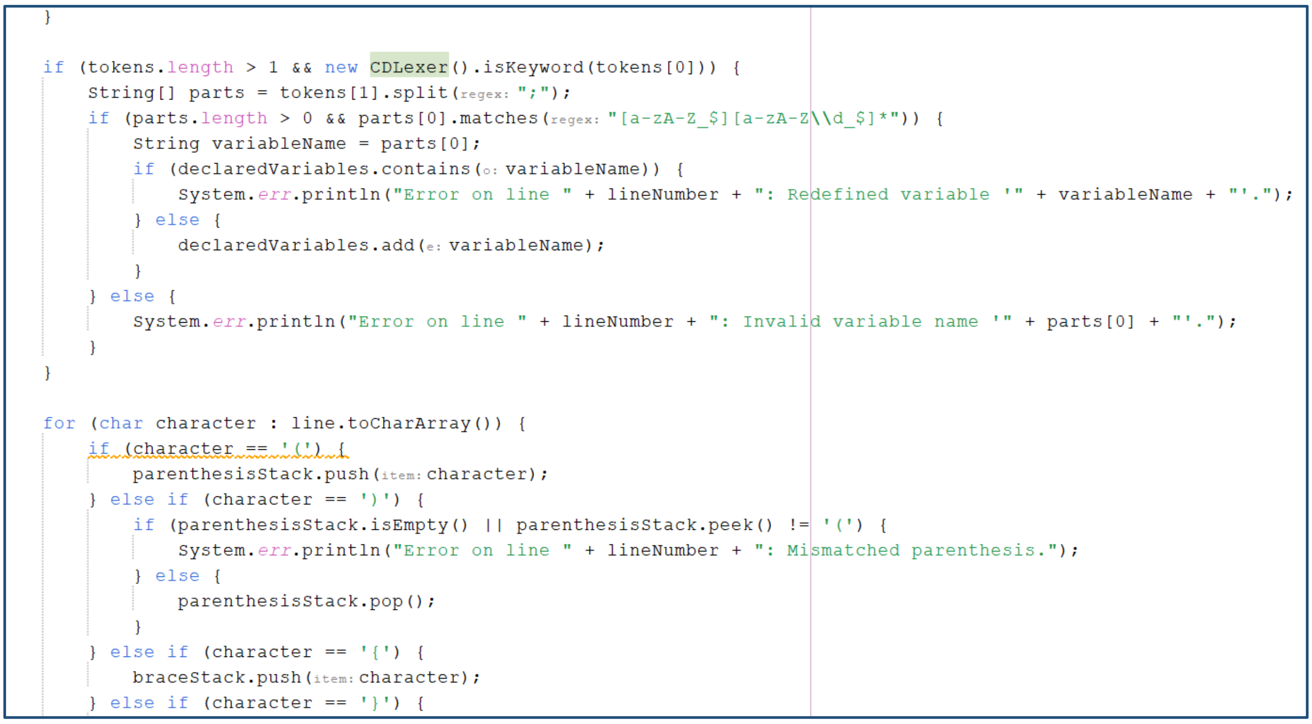


```
    public String identifyTokenType(String token) {
        if (TOKEN.containsKey(key:token)) {
            return TOKEN.get(key:token);
        } else if (token.matches(regex: "[0-9]+")) {
            return "integer_literal";
        } else if (token.matches(regex: "[0-9]*\\.[0-9]+")) {
            return "floating_point_literal";
        } else if (token.matches(regex: "\".*\"")) {
            return "string_literal";
        } else if (token.matches(regex: "[a-zA-Z_$][a-zA-Z\\d_$]*")) {
            return "identifier";
        } else {
            return "unknown";
        }
    }

    public boolean isKeyword(String token) {
        return TOKEN.containsKey(key:token) && "keyword".equals(anObject: TOKEN.get(key:token));
    }
}
```

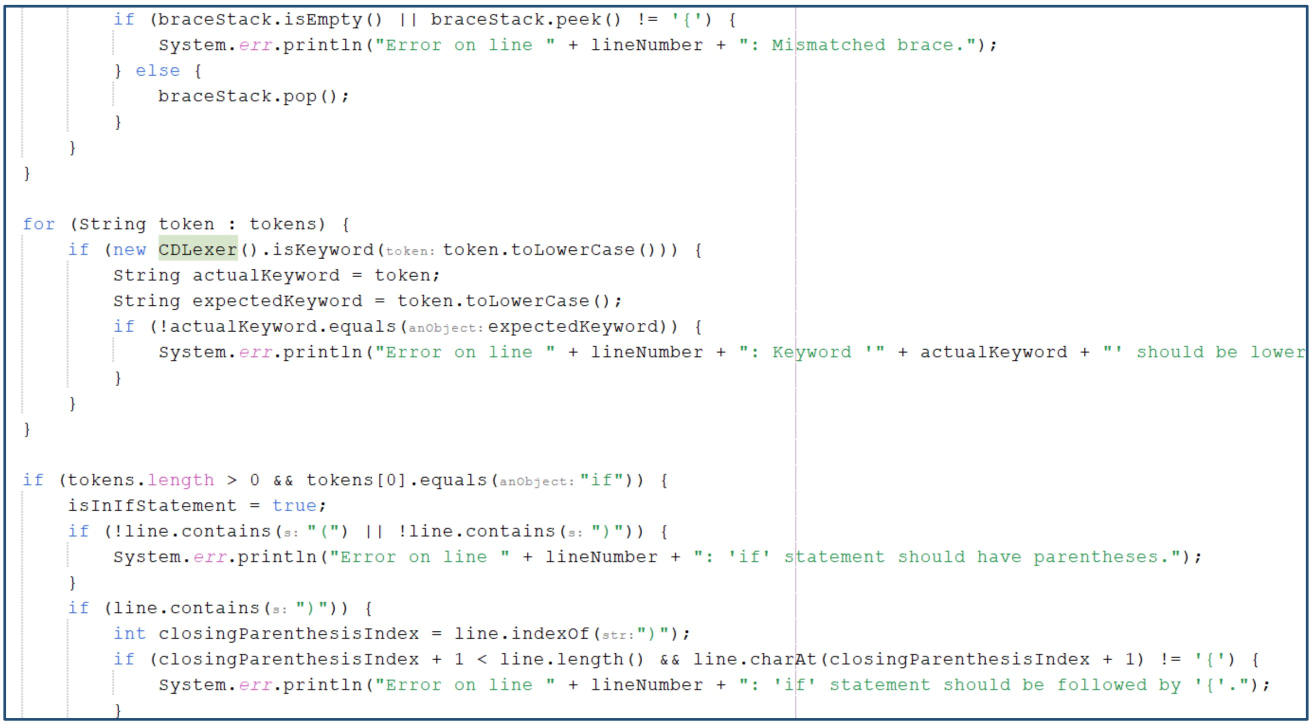
3- Code Parser:



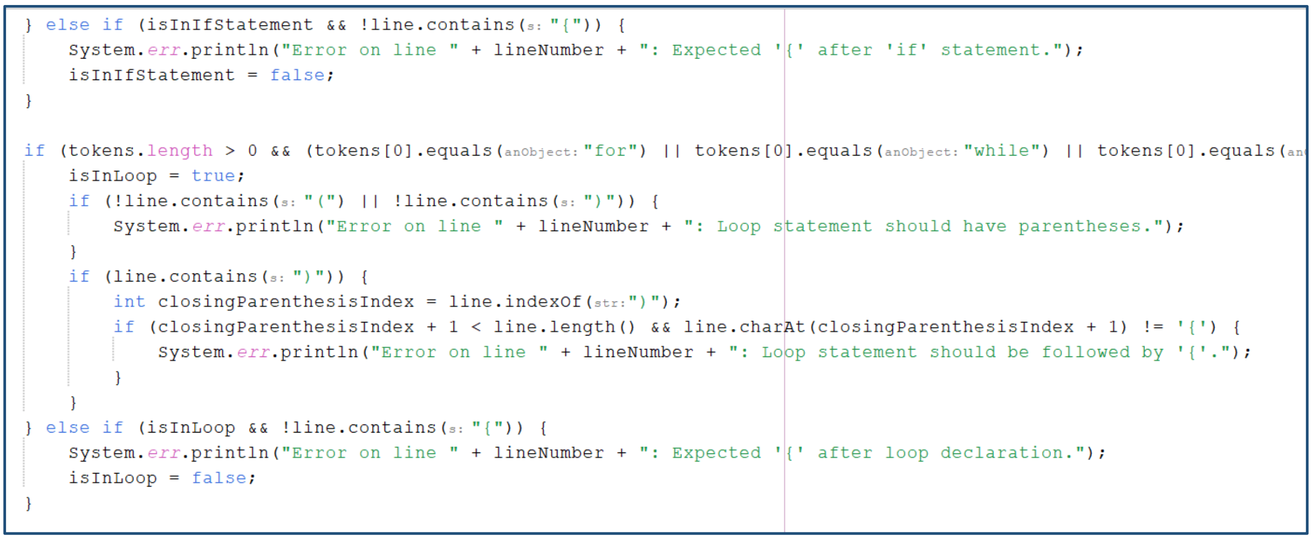
```java
package compilerproject;
import java.io.*;
import java.util.*;
public class CDParser {
    public void analyzeTokens(String inputFilePath, Set<String> declaredVariables) throws IOException {
        Scanner fileScanner = new Scanner(new File(pathname: inputFilePath));

        Stack<Character> parenthesisStack = new Stack<>();
        Stack<Character> braceStack = new Stack<>();
        int lineNumber = 1;
        boolean isInIfStatement = false;
        boolean isInLoop = false;

        while (fileScanner.hasNextLine()) {
            String line = fileScanner.nextLine().trim();
            String[] tokens = line.split(regex: "\\s+");

            if (line.isEmpty()) {
                lineNumber++;
                continue;
            }

            if (!line.endsWith(suffix: "{") && !line.endsWith(suffix: "}") && !line.endsWith(suffix: ";") && !line.endsWit
                System.err.println("Error on line " + lineNumber + ": Missing semicolon.");
            }

            if (tokens.length > 1 && new CDLexer().isKeyword(tokens[0])) {
                String[] parts = tokens[1].split(regex: ";");
```



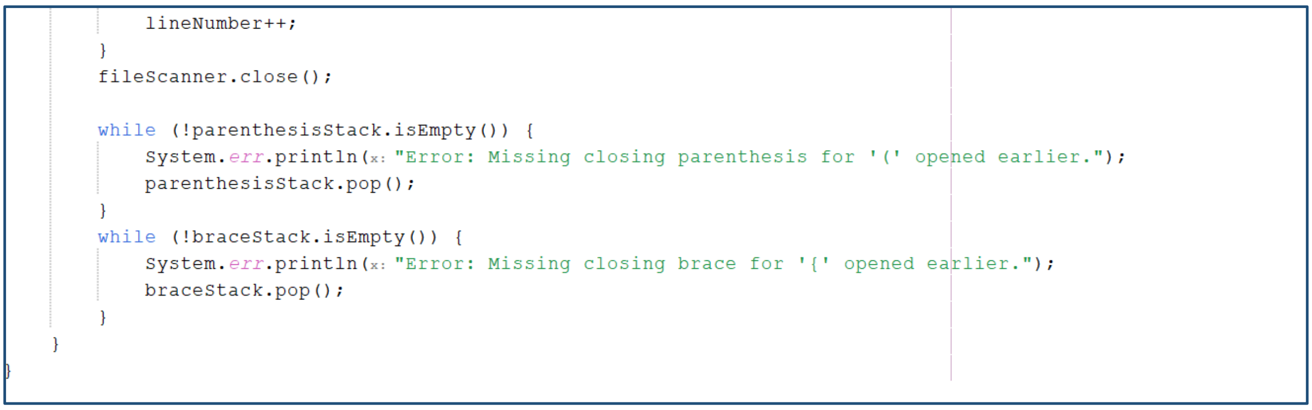
```java
}

if (tokens.length > 1 && new CDLexer().isKeyword(tokens[0])) {
    String[] parts = tokens[1].split(regex: ";");
    if (parts.length > 0 && parts[0].matches(regex: "[a-zA-Z_$][a-zA-Z\\d_$]*")) {
        String variableName = parts[0];
        if (declaredVariables.contains(o: variableName)) {
            System.err.println("Error on line " + lineNumber + ": Redefined variable '" + variableName + "'.");
        } else {
            declaredVariables.add(e: variableName);
        }
    } else {
        System.err.println("Error on line " + lineNumber + ": Invalid variable name '" + parts[0] + "'.");
    }
}

for (char character : line.toCharArray()) {
    if (character == '(') {
        parenthesisStack.push(item: character);
    } else if (character == ')') {
        if (parenthesisStack.isEmpty() || parenthesisStack.peek() != '(') {
            System.err.println("Error on line " + lineNumber + ": Mismatched parenthesis.");
        } else {
            parenthesisStack.pop();
        }
    } else if (character == '{') {
        braceStack.push(item: character);
    } else if (character == '}') {
```



```
        if (braceStack.isEmpty() || braceStack.peek() != '{') {
            System.err.println("Error on line " + lineNumber + ": Mismatched brace.");
        } else {
            braceStack.pop();
        }
    }
}

for (String token : tokens) {
    if (new CDLexer().isKeyword(token: token.toLowerCase())) {
        String actualKeyword = token;
        String expectedKeyword = token.toLowerCase();
        if (!actualKeyword.equals(anObject: expectedKeyword)) {
            System.err.println("Error on line " + lineNumber + ": Keyword '" + actualKeyword + "' should be lower
        }
    }
}

if (tokens.length > 0 && tokens[0].equals(anObject: "if")) {
    isInIfStatement = true;
    if (!line.contains(s: "(") || !line.contains(s: ")")) {
        System.err.println("Error on line " + lineNumber + ": 'if' statement should have parentheses.");
    }
    if (line.contains(s: ")")) {
        int closingParenthesisIndex = line.indexOf(str: ")");
        if (closingParenthesisIndex + 1 < line.length() && line.charAt(closingParenthesisIndex + 1) != '{') {
            System.err.println("Error on line " + lineNumber + ": 'if' statement should be followed by '{'.");
        }
```

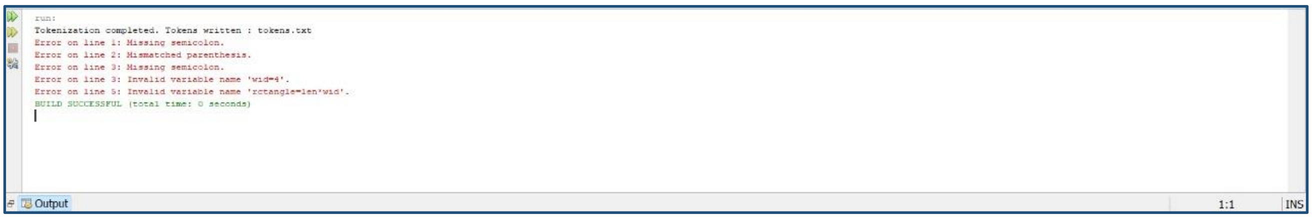


```
} else if (isInIfStatement && !line.contains(s: "{")) {
    System.err.println("Error on line " + lineNumber + ": Expected '{' after 'if' statement.");
    isInIfStatement = false;
}

if (tokens.length > 0 && (tokens[0].equals(anObject: "for") || tokens[0].equals(anObject: "while") || tokens[0].equals(an
    isInLoop = true;
    if (!line.contains(s: "(") || !line.contains(s: ")")) {
        System.err.println("Error on line " + lineNumber + ": Loop statement should have parentheses.");
    }
    if (line.contains(s: ")")) {
        int closingParenthesisIndex = line.indexOf(str: ")");
        if (closingParenthesisIndex + 1 < line.length() && line.charAt(closingParenthesisIndex + 1) != '{') {
            System.err.println("Error on line " + lineNumber + ": Loop statement should be followed by '{'.");
        }
    }
} else if (isInLoop && !line.contains(s: "{")) {
    System.err.println("Error on line " + lineNumber + ": Expected '{' after loop declaration.");
    isInLoop = false;
}
```



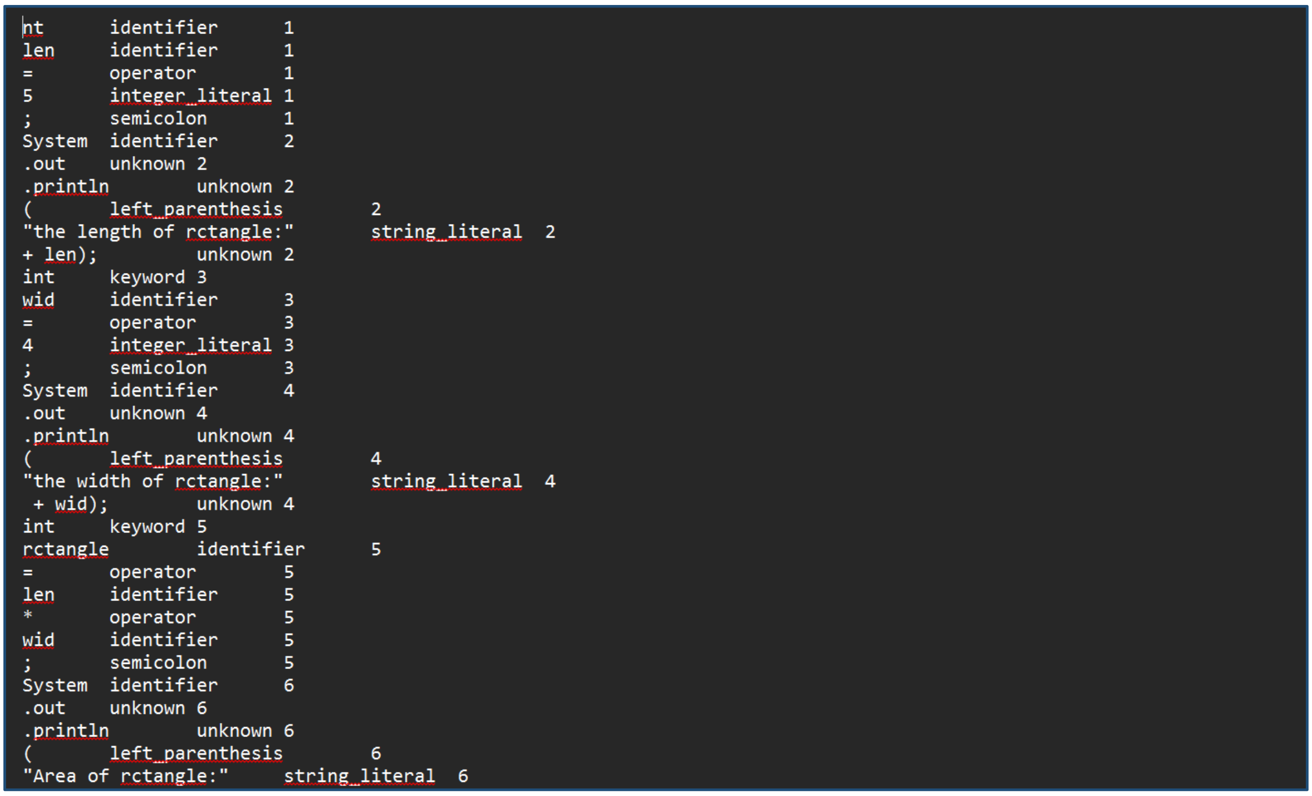
```
            lineNumber++;
        }
        fileScanner.close();

        while (!parenthesisStack.isEmpty()) {
            System.err.println(x: "Error: Missing closing parenthesis for '(' opened earlier.");
            parenthesisStack.pop();
        }
        while (!braceStack.isEmpty()) {
            System.err.println(x: "Error: Missing closing brace for '{' opened earlier.");
            braceStack.pop();
        }
    }
}
```

Output of code:



```
run:
Tokenization completed. Tokens written : tokens.txt
Error on line 1: Missing semicolon.
Error on line 2: Mismatched parenthesis.
Error on line 3: Missing semicolon.
Error on line 3: Invalid variable name 'wid=4'.
Error on line 5: Invalid variable name 'rctangle=len*wid'.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tokens of code:



```
nt      identifier      1
len     identifier      1
=       operator        1
5       integer_literal 1
;       semicolon       1
System  identifier      2
.out    unknown 2
.println        unknown 2
(       left_parenthesis        2
"the length of rctangle:"       string_literal  2
+ len);         unknown 2
int     keyword 3
wid     identifier      3
=       operator        3
4       integer_literal 3
;       semicolon       3
System  identifier      4
.out    unknown 4
.println        unknown 4
(       left_parenthesis        4
"the width of rctangle:"        string_literal  4
 + wid);        unknown 4
int     keyword 5
rctangle        identifier      5
=       operator        5
len     identifier      5
*       operator        5
wid     identifier      5
;       semicolon       5
System  identifier      6
.out    unknown 6
.println        unknown 6
(       left_parenthesis        6
"Area of rctangle:"     string_literal  6
```

Reference:

[1] https://github.com/welovelain/Java-JFlex-Cup-Example

[2] https://github.com/YisusTecFBI/Compiler/

[3] https://github.com/IvanoBilenchi/jflex-cup-example

[4] https://github.com/IcedGarion/jcup-jflex

[5] https://github.com/nguyendinhnien/Jflex-project/tree/master/JflexAssignment

[6] https://github.com/ragalayaswara/jflexrepo

[7] https://github.com/RenatusRS/Mikrojava-Compiler

[8] https://lms.uqu.edu.sa/bbcswebdav/pid-4020231-dt-content-rid-124114478_1/xid-124114478_1

[9] https://youtu.be/bE6FQH7lqbo?si=VXMXBAvmqe91a_3n

[10] https://youtu.be/nZfovY1KoPo?si=id2DvMQaZAPxHMSS

[11] https://youtu.be/Ro1-hr_e2Es?list=PLPSFnlxEu99ENrSX4yYAlAnezOSuZEvkb

[12] https://youtu.be/ny85GdeERTg

[13]
https://youtu.be/D61GG6BzD8M?list=PLPSFnlxEu99ENrSX4yYAlAnezOSuZEvkb

[14]
https://youtu.be/pUNe8oBgFZI?list=PLPSFnlxEu99ENrSX4yYAlAnezOSuZEvkb