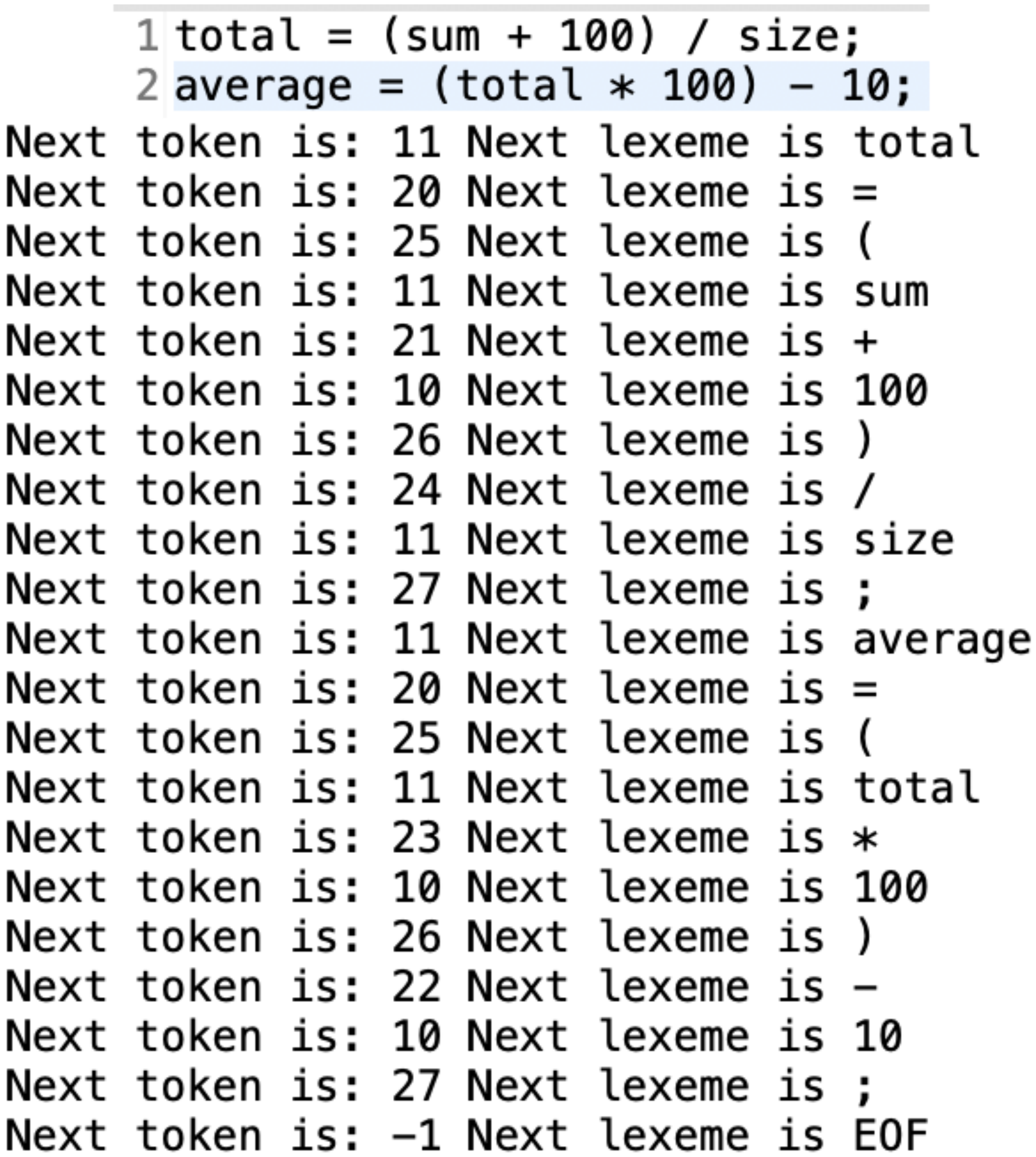
CMPSCI4250 Project1

9/20/19

```java
package edu.umsl.cs4250.project1;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class Front {
    private static final int MAX_LEXEME_LEN = 100;
    private static final int EOF = -1;

    /* Character classes */
    private static final int LETTER = 0;
    private static final int DIGIT = 1;
    private static final int UNKNOWN = 99;

    /* Token codes */
    private static final int INT_LIT = 10;
    private static final int IDENT = 11;
    private static final int ASSIGN_OP = 20;
    private static final int ADD_OP = 21;
    private static final int SUB_OP = 22;
    private static final int MULT_OP = 23;
    private static final int DIV_OP = 24;
    private static final int LEFT_PAREN = 25;
    private static final int RIGHT_PAREN = 26;
    private static final int SEMIC = 27;

    /* Global declarations */
    /* Variables */
    private static int charClass;
    private static char lexeme[];
    private static char nextChar;
    private static int lexLen;
    private static int nextToken;
    private static File file;
    private static FileInputStream fis;

    /*
         * lookup - a function to lookup operators, parentheses, and terminators and return the token
         */
    public static int lookup(char ch) {
        switch (ch) {
        case '=':
            addChar();
            nextToken = ASSIGN_OP;
            break;
        case '(':
            addChar();
            nextToken = LEFT_PAREN;
            break;
        case ')':
            addChar();
            nextToken = RIGHT_PAREN;
            break;
        case '+':
            addChar();
            nextToken = ADD_OP;
            break;
        case '-':
            addChar();
            nextToken = SUB_OP;
            break;
        case '*':
            addChar();
            nextToken = MULT_OP;
            break;
        case '/':
            addChar();
            nextToken = DIV_OP;
            break;
        case ';':
            addChar();
            nextToken = SEMIC;
            break;
        default:
            addChar();
            nextToken = EOF;
            break;
        }

        return nextToken;
    }

    /* addChar - a function to add nextChar to lexeme */
    public static void addChar() {
        if (lexLen < MAX_LEXEME_LEN) {
            lexeme[lexLen++] = nextChar;
        } else {
            System.out.flush();
            System.err.flush();
            System.err.println("Error - lexeme is too long");
        }
    }

    /*
         * getChar - a function to get the next character of input and determine its
         * character class
         */
    public static void getChar() {
        try {
            if (fis.available() > 0) {
                do {
                    nextChar = (char) fis.read();
                } while (nextChar == '\n' || nextChar == '\r');

                if (Character.isLetter(nextChar)) {
                    charClass = LETTER;
                } else if (Character.isDigit(nextChar)) {
                    charClass = DIGIT;
                } else {
                    charClass = UNKNOWN;
                }
            } else {
                charClass = EOF;
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /*
         * getNonBlank - a function to call getChar until it returns a non-whitespace
         * character
         */
    public static void getNonBlank() {
        while (Character.isSpaceChar(nextChar)) {
            getChar();
        }
    }

    /* lex - a simple lexical analyzer for arithmetic expressions */
    public static int lex() {
        lexLen = 0;
        getNonBlank();
        switch (charClass) {
        case LETTER:
            addChar();
            getChar();
            while (charClass == LETTER || charClass == DIGIT) {
                addChar();
                getChar();
            }
            nextToken = IDENT;
            break;
        case DIGIT:
            addChar();
            getChar();
            while (charClass == DIGIT) {
                addChar();
                getChar();
            }
            nextToken = INT_LIT;
            break;
        case UNKNOWN:
            lookup(nextChar);
            getChar();
            break;
        case EOF:
            lexLen = 3;
            lexeme[0] = 'E';
            lexeme[1] = 'O';
            lexeme[2] = 'F';
            nextToken = EOF;
            break;
        }

        System.out.flush();
        System.out.print("Next token is: " + nextToken + " Next lexeme is ");
        for (int i = 0; i < lexLen; i++) {
            System.out.print(lexeme[i]);
        }
        System.out.println();

        return nextToken;
    }

    /* main driver */
    public static void main(String args[]) {
        lexLen = 0;
        lexeme = new char[MAX_LEXEME_LEN + 2];
        for (int i = 0; i < lexeme.length; i++) {
            lexeme[i] = 0;
        }

        file = new File("res/input.txt");
        if (!file.exists()) {
            System.err.println(file.getName() + " does not exist.");
            return;
        }
        if (!(file.isFile() && file.canRead())) {
            System.err.println(file.getName() + " cannot be read from.");
            return;
        }

        try {
            fis = new FileInputStream(file);
            getChar();
            do {
                lex();
            } while (nextToken != EOF);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

PDF document made with CodePrint using Prism



```
total = (sum + 100) / size;
average = (total * 100) - 10;
```

```
Next token is: 11 Next lexeme is total
Next token is: 20 Next lexeme is =
Next token is: 25 Next lexeme is (
Next token is: 11 Next lexeme is sum
Next token is: 21 Next lexeme is +
Next token is: 10 Next lexeme is 100
Next token is: 26 Next lexeme is )
Next token is: 24 Next lexeme is /
Next token is: 11 Next lexeme is size
Next token is: 27 Next lexeme is ;
Next token is: 11 Next lexeme is average
Next token is: 20 Next lexeme is =
Next token is: 25 Next lexeme is (
Next token is: 11 Next lexeme is total
Next token is: 23 Next lexeme is *
Next token is: 10 Next lexeme is 100
Next token is: 26 Next lexeme is )
Next token is: 22 Next lexeme is -
Next token is: 10 Next lexeme is 10
Next token is: 27 Next lexeme is ;
Next token is: -1 Next lexeme is EOF
```