Compiler project

هاجر محمود عبد الله	444015474	Writing in parser code
شيماء لافي الزبيدي	444003238	writing main method part ,tokens part
لطيفة يحيى الذيبي	444014761	Writing in parser code
ريم عقيل الصالحي	444014767	Writing scanner code

Overall we've helped each other in the project

Introduction:

Compilers are very important tools for compiling the high level language into low level language like machine code.

This transformation is fundamental since the computer doesn't understand high level languages written by humans.

Compilation operation goes into lots of phases included in analysis and synthesis parts:

Analysis:

lexical analysis / which reads the source code and translates it into their related tokens.

Syntax analysis/ now the parser arrange the produced tokens into a syntax tree ensuring that tokens follow grammar rules.

semantic analysis/ which involves checking the produced syntax tree for semantic errors.

Synthesis:

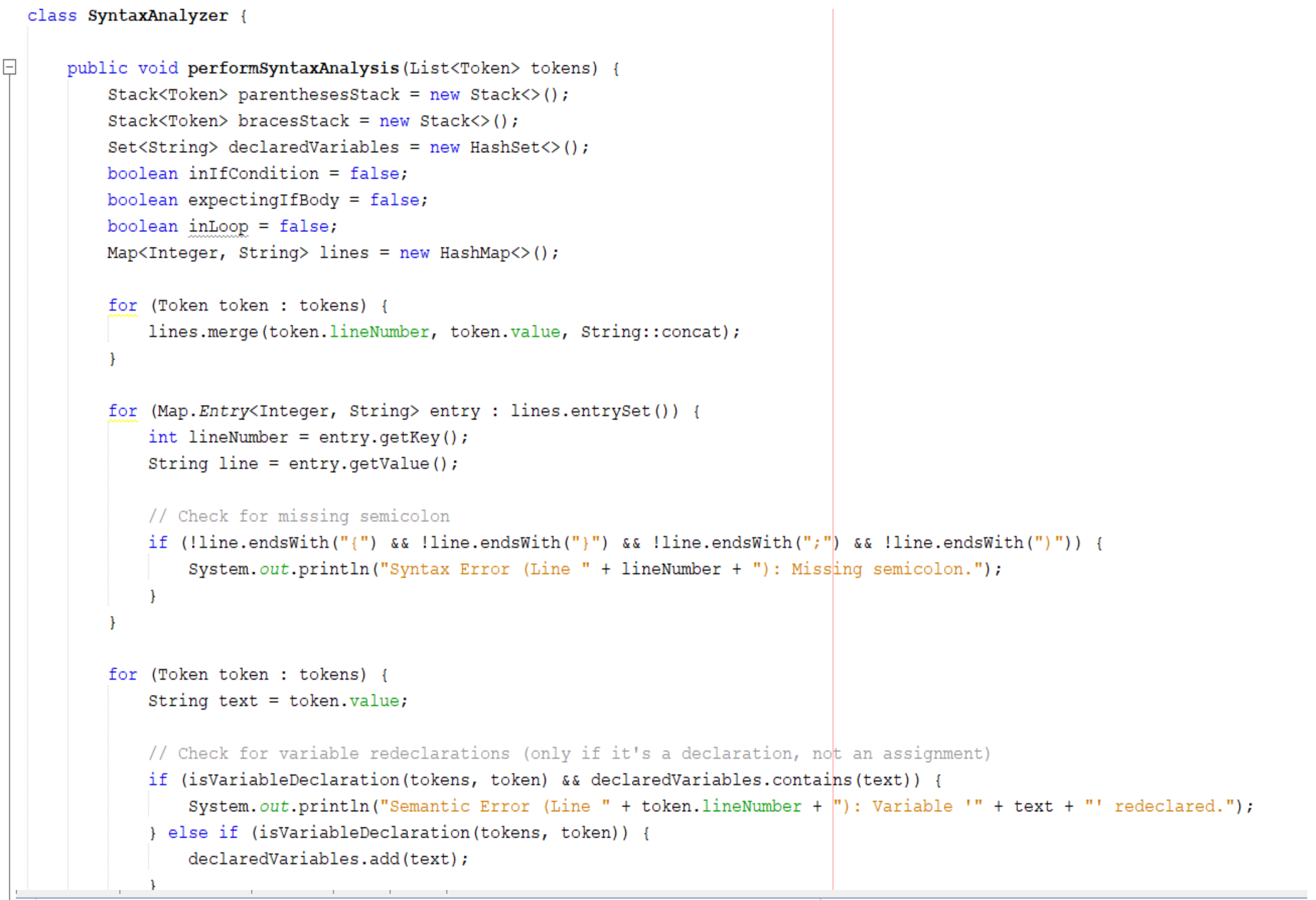
Intermediate code generation / the compiler translates the syntax tree into a lower level code , serves as a bridge between high level language and the machine code.

Optimization /now some improvements are done for the intermediate code to improve performance and efficiency.

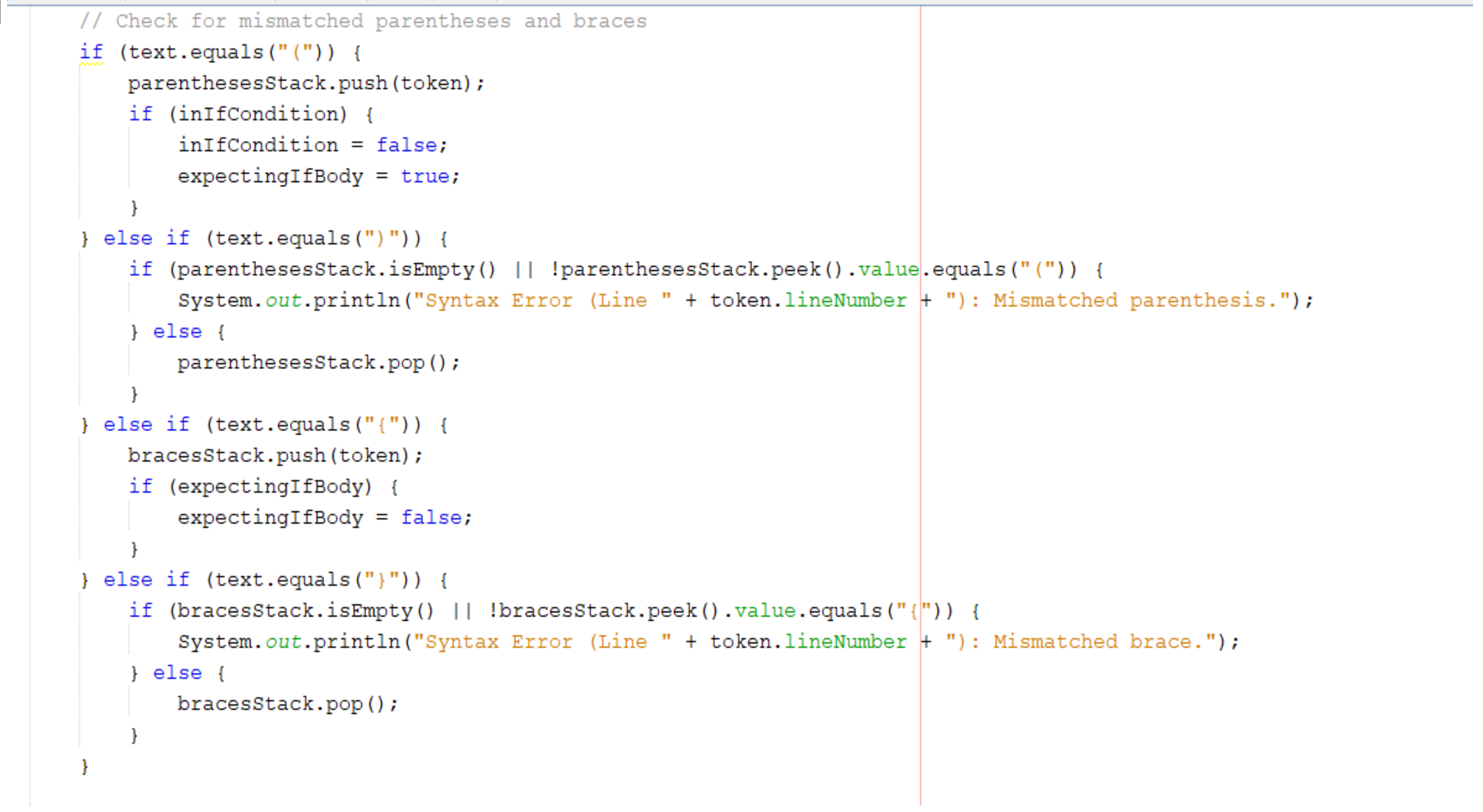
Code generation/ the final phase in which the optimized code is translated into machine code.

And in this project we've created a simple compiler for simple inputs.

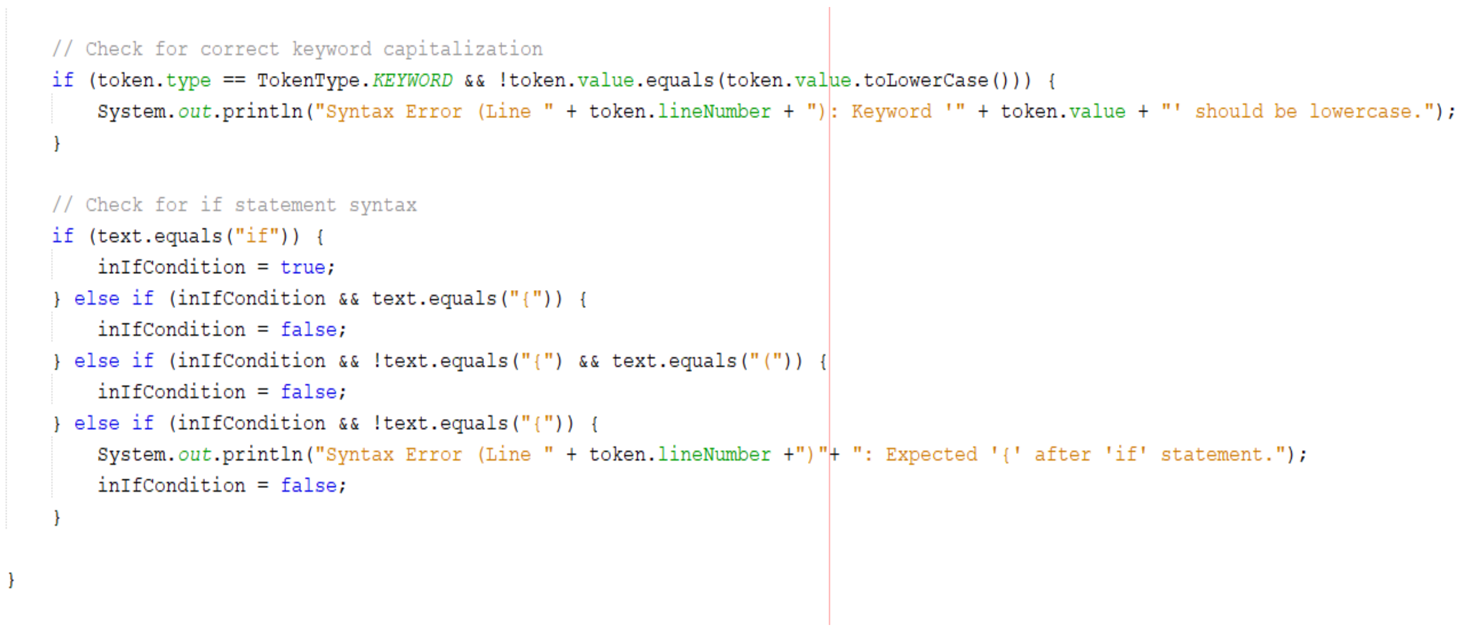
Testing code parts:



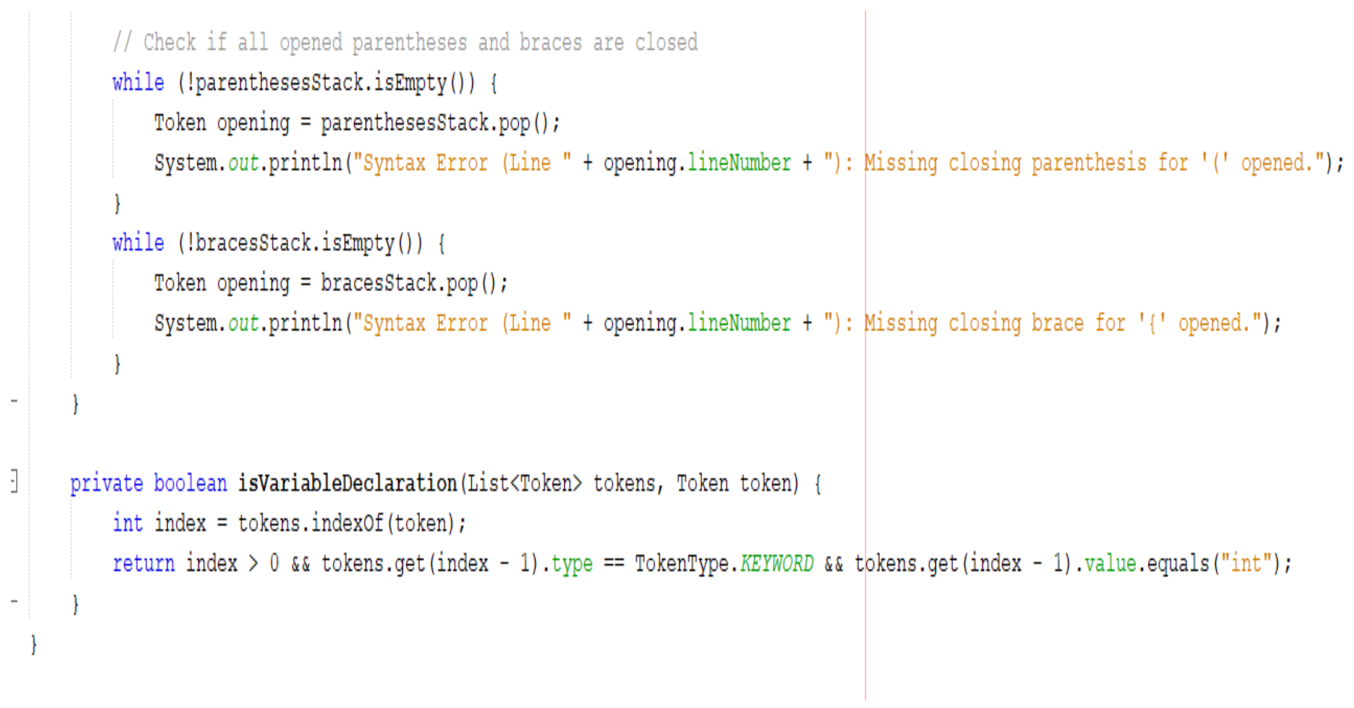
```java
class SyntaxAnalyzer {

    public void performSyntaxAnalysis(List<Token> tokens) {
        Stack<Token> parenthesesStack = new Stack<>();
        Stack<Token> bracesStack = new Stack<>();
        Set<String> declaredVariables = new HashSet<>();
        boolean inIfCondition = false;
        boolean expectingIfBody = false;
        boolean inLoop = false;
        Map<Integer, String> lines = new HashMap<>();

        for (Token token : tokens) {
            lines.merge(token.lineNumber, token.value, String::concat);
        }

        for (Map.Entry<Integer, String> entry : lines.entrySet()) {
            int lineNumber = entry.getKey();
            String line = entry.getValue();

            // Check for missing semicolon
            if (!line.endsWith("{") && !line.endsWith("}") && !line.endsWith(";") && !line.endsWith(")")) {
                System.out.println("Syntax Error (Line " + lineNumber + "): Missing semicolon.");
            }
        }

        for (Token token : tokens) {
            String text = token.value;

            // Check for variable redeclarations (only if it's a declaration, not an assignment)
            if (isVariableDeclaration(tokens, token) && declaredVariables.contains(text)) {
                System.out.println("Semantic Error (Line " + token.lineNumber + "): Variable '" + text + "' redeclared.");
            } else if (isVariableDeclaration(tokens, token)) {
                declaredVariables.add(text);
            }
```



```java
            // Check for mismatched parentheses and braces
            if (text.equals("(")) {
                parenthesesStack.push(token);
                if (inIfCondition) {
                    inIfCondition = false;
                    expectingIfBody = true;
                }
            } else if (text.equals(")")) {
                if (parenthesesStack.isEmpty() || !parenthesesStack.peek().value.equals("(")) {
                    System.out.println("Syntax Error (Line " + token.lineNumber + "): Mismatched parenthesis.");
                } else {
                    parenthesesStack.pop();
                }
            } else if (text.equals("{")) {
                bracesStack.push(token);
                if (expectingIfBody) {
                    expectingIfBody = false;
                }
            } else if (text.equals("}")) {
                if (bracesStack.isEmpty() || !bracesStack.peek().value.equals("{")) {
                    System.out.println("Syntax Error (Line " + token.lineNumber + "): Mismatched brace.");
                } else {
                    bracesStack.pop();
                }
            }
```

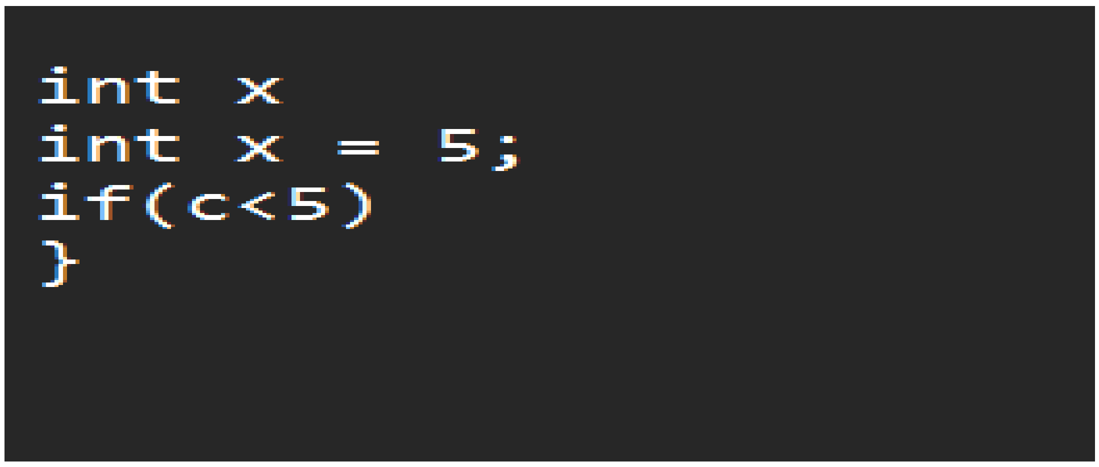


```
    // Check for correct keyword capitalization
    if (token.type == TokenType.KEYWORD && !token.value.equals(token.value.toLowerCase())) {
        System.out.println("Syntax Error (Line " + token.lineNumber + "): Keyword '" + token.value + "' should be lowercase.");
    }

    // Check for if statement syntax
    if (text.equals("if")) {
        inIfCondition = true;
    } else if (inIfCondition && text.equals("{")) {
        inIfCondition = false;
    } else if (inIfCondition && !text.equals("{") && text.equals("(")) {
        inIfCondition = false;
    } else if (inIfCondition && !text.equals("{")) {
        System.out.println("Syntax Error (Line " + token.lineNumber +")"+ ": Expected '{' after 'if' statement.");
        inIfCondition = false;
    }

}
```



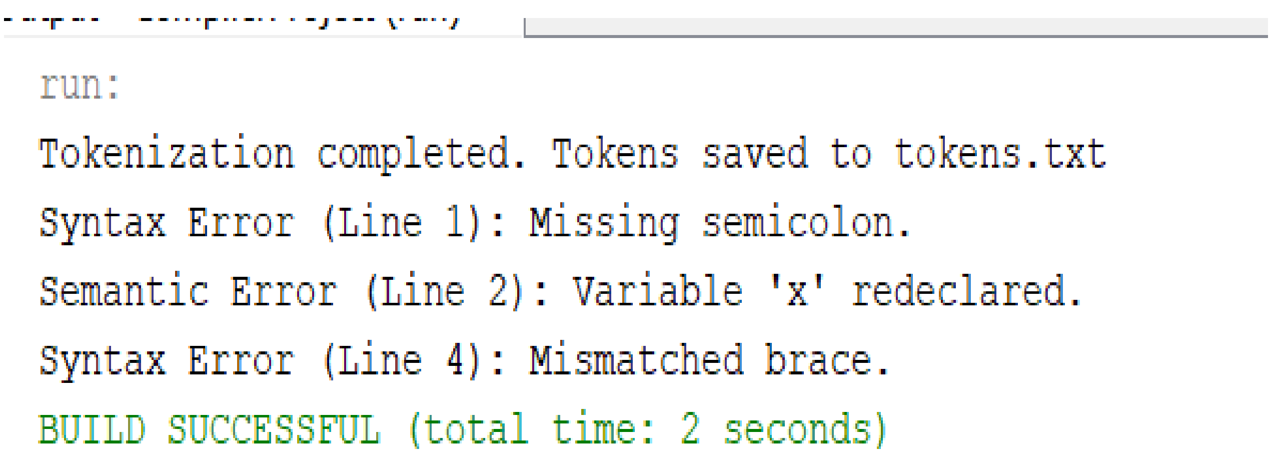
```
        // Check if all opened parentheses and braces are closed
        while (!parenthesesStack.isEmpty()) {
            Token opening = parenthesesStack.pop();
            System.out.println("Syntax Error (Line " + opening.lineNumber + "): Missing closing parenthesis for '(' opened.");
        }
        while (!bracesStack.isEmpty()) {
            Token opening = bracesStack.pop();
            System.out.println("Syntax Error (Line " + opening.lineNumber + "): Missing closing brace for '{' opened.");
        }
    }

    private boolean isVariableDeclaration(List<Token> tokens, Token token) {
        int index = tokens.indexOf(token);
        return index > 0 && tokens.get(index - 1).type == TokenType.KEYWORD && tokens.get(index - 1).value.equals("int");
    }
}
```

This is correct since the input is :



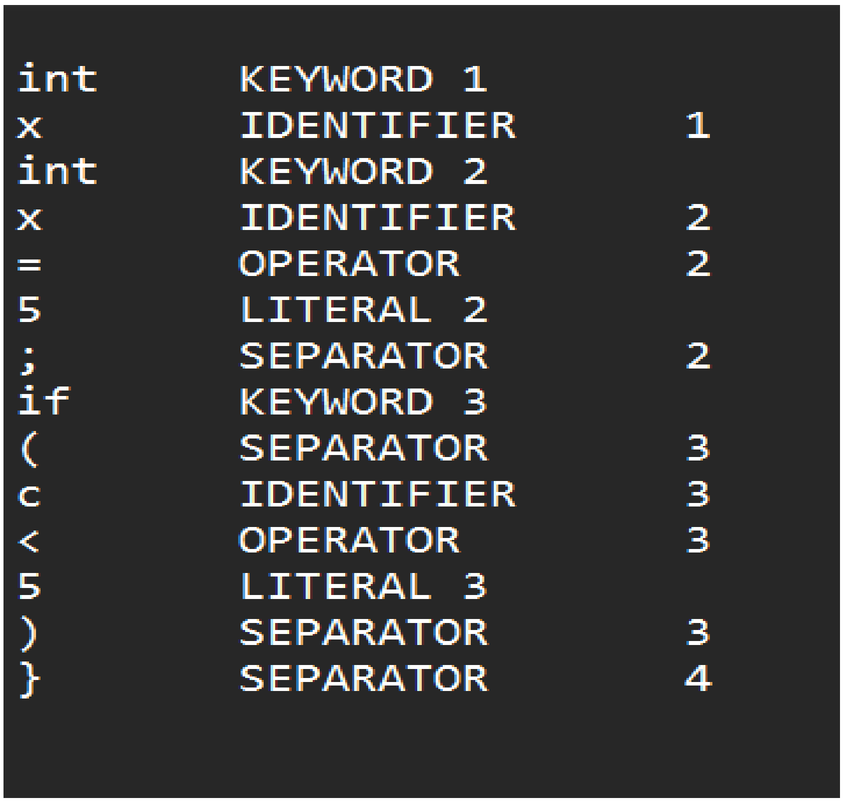
```
int x
int x = 5;
if(c<5)
}
```

And the result showed that there is a missing semicolon, a redeclaration for (x) and a mismatched brace.



```
run:
Tokenization completed. Tokens saved to tokens.txt
Syntax Error (Line 1): Missing semicolon.
Semantic Error (Line 2): Variable 'x' redeclared.
Syntax Error (Line 4): Mismatched brace.
BUILD SUCCESSFUL (total time: 2 seconds)
```

also insuring that tokens have been written for the appropriate inputs :



```
int      KEYWORD 1
x        IDENTIFIER     1
int      KEYWORD 2
x        IDENTIFIER     2
=        OPERATOR       2
5        LITERAL 2
;        SEPARATOR      2
if       KEYWORD 3
(        SEPARATOR      3
c        IDENTIFIER     3
<        OPERATOR       3
5        LITERAL 3
)        SEPARATOR      3
}        SEPARATOR      4
```

So that was our project .

Resources used:

Geeks for geeks

Compilers -Principles ,techniques and tools

And for software:

NetBeans

Virtual studio code