Lab 4 Documentation

Komsa Attila-Janos, 934/2

Github link: https://github.com/kms77/FLCD/tree/main/Lab3

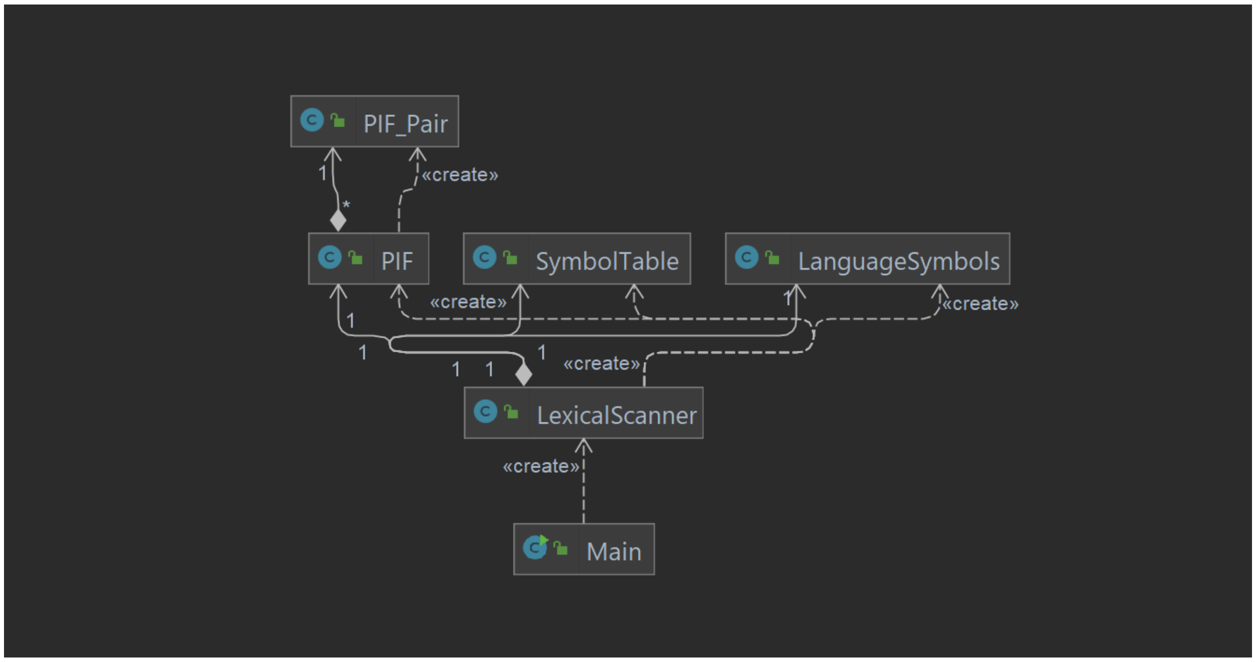
Updated documentation link:
https://github.com/kms77/FLCD/tree/main/Documentation

Statement: Implement a scanner (lexical analyzer): Implement the scanning algorithm and use ST from lab 2 for the symbol table.

Input: Programs p1/p2/p3/p1err and token.in (see Lab 1a)

Output: PIF.out, ST.out, message “lexically correct” or “lexical error + location”

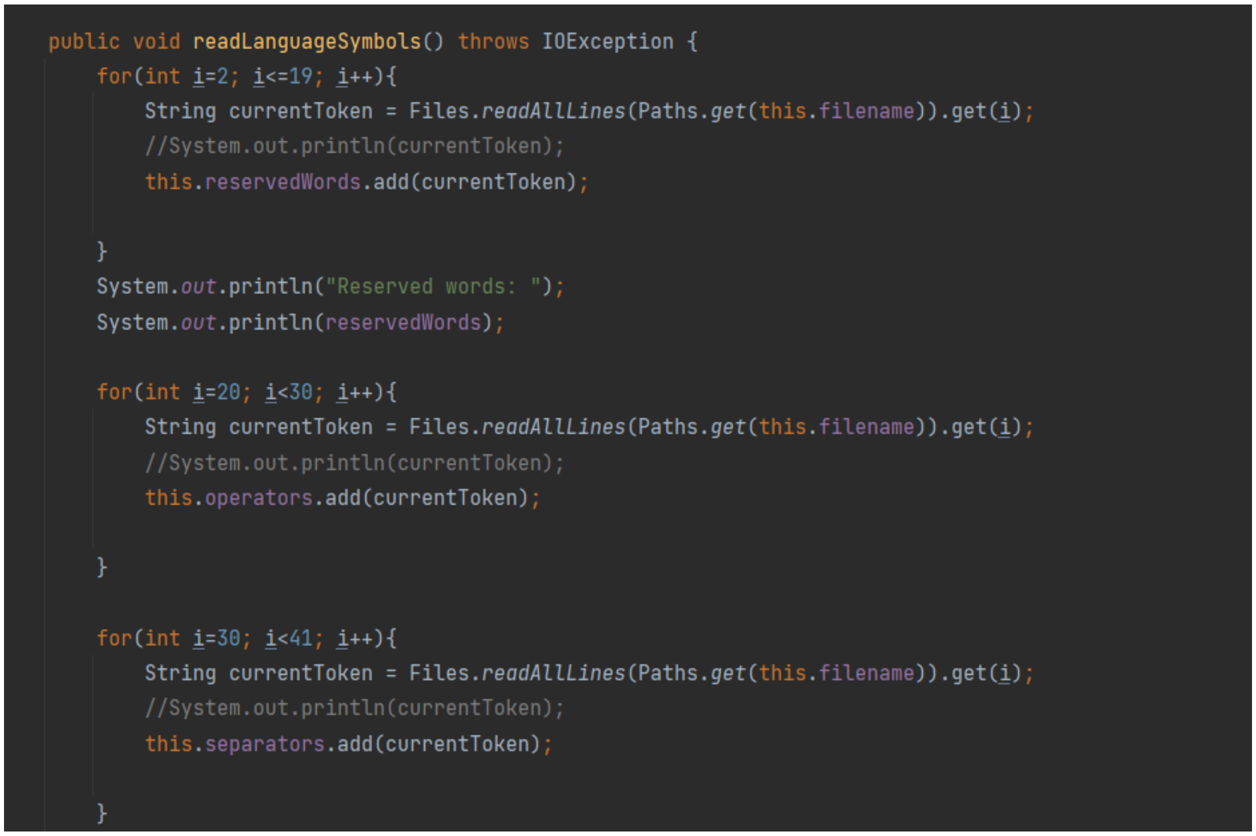
In order to implement the scanner, I have used the classes: SymbolTable, LanguageSymbols, PIF_Pair, PIF and LexicalScanner. The class diagram with the dependencies is the following:



SymbolTable

The SymbolTable is implemented as a HashTable and has the addElement and searchElement functionalities, which run in O(1).

In the SymbolTable we are going to add only identifiers and constants. The rest of the tokens (separators, operators and reserved words) are going to be kept in separate lists by the LanguageSymbols class, which reads all the tokens from the token.in file and classifies them:

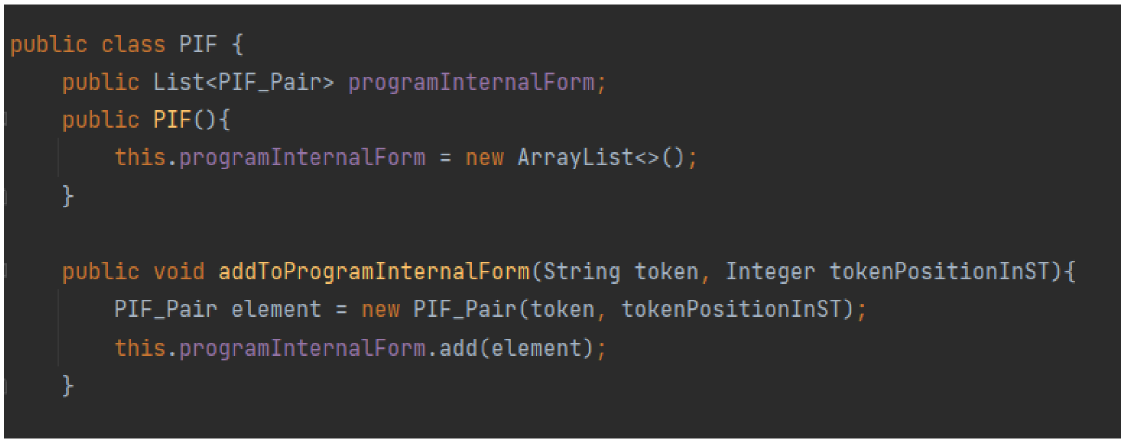


```java
public void readLanguageSymbols() throws IOException {
    for(int i=2; i<=19; i++){
        String currentToken = Files.readAllLines(Paths.get(this.filename)).get(i);
        //System.out.println(currentToken);
        this.reservedWords.add(currentToken);

    }
    System.out.println("Reserved words: ");
    System.out.println(reservedWords);

    for(int i=20; i<30; i++){
        String currentToken = Files.readAllLines(Paths.get(this.filename)).get(i);
        //System.out.println(currentToken);
        this.operators.add(currentToken);

    }

    for(int i=30; i<41; i++){
        String currentToken = Files.readAllLines(Paths.get(this.filename)).get(i);
        //System.out.println(currentToken);
        this.separators.add(currentToken);

    }
```

PIF

The Program Internal Form (PIF) is based on two classes:

- PIF_Pair - consists of a (token, positionInST) tuple, positionInST represents the position of the given token in the Symbol Table. If the token is not in the Symbol Table, its position on the ST will be -1
- PIF- kept as a Java List with elements of type PIF_Pair. It has the add and print functionalities implemented.



```java
public class PIF {
    public List<PIF_Pair> programInternalForm;
    public PIF(){
        this.programInternalForm = new ArrayList<>();
    }

    public void addToProgramInternalForm(String token, Integer tokenPositionInST){
        PIF_Pair element = new PIF_Pair(token, tokenPositionInST);
        this.programInternalForm.add(element);
    }
```

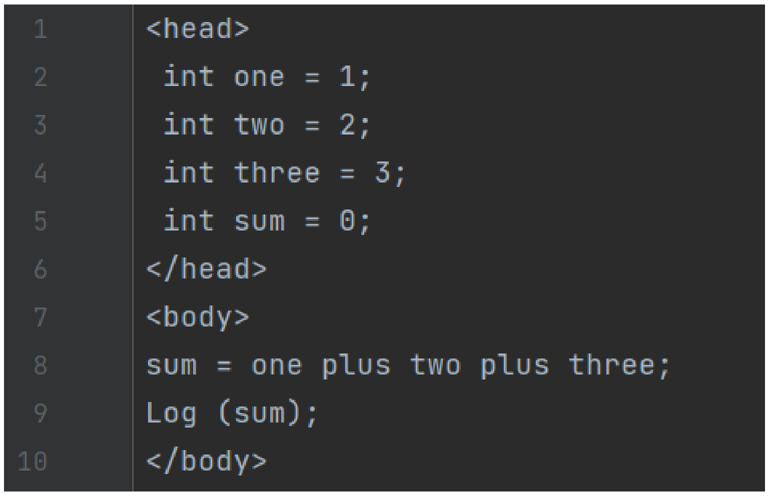
Scanner

The scanning part is done by the functions:

- parseLine() - takes as an input a line from the file, which is of type String, and splits it into tokens. For each token, it checks its type and, if it's an identifier or constant, adds it to the Symbol Table, and then to PIF. If a token cannot be classified, a lexical error message, along with the line where it is found, will be printed
- parseFile() – scans the lines of the program file and calls the above mentioned method to each of them, in order to split all the program into tokens. It also writes the results to the ST.out and PIF.out files.

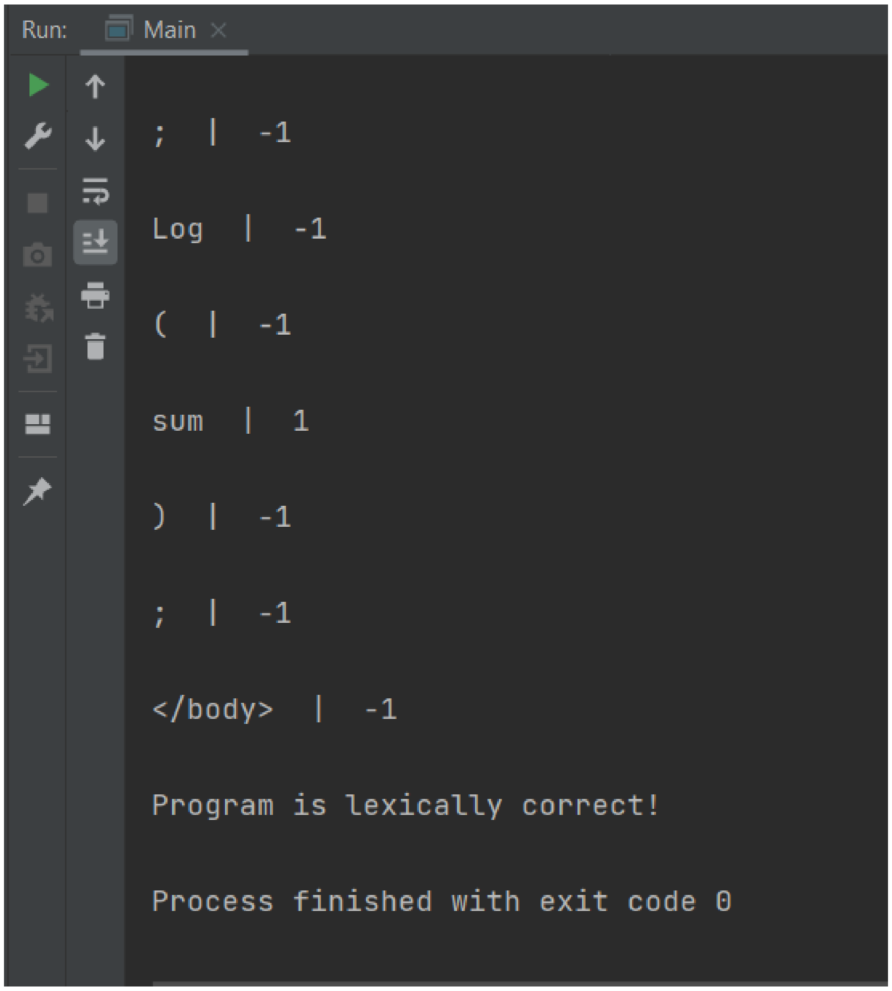
Tests

p1.txt:



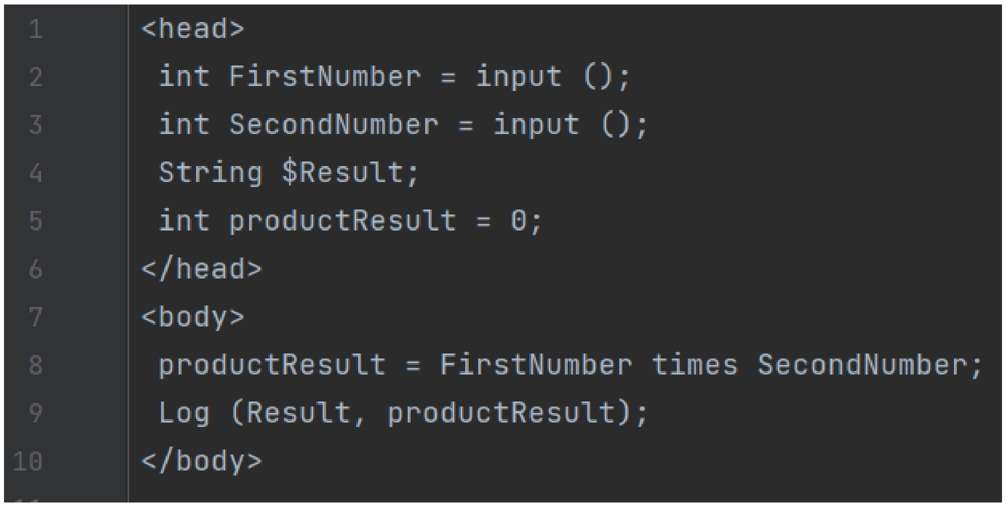
```
<head>
 int one = 1;
 int two = 2;
 int three = 3;
 int sum = 0;
</head>
<body>
sum = one plus two plus three;
Log (sum);
</body>
```

Output:



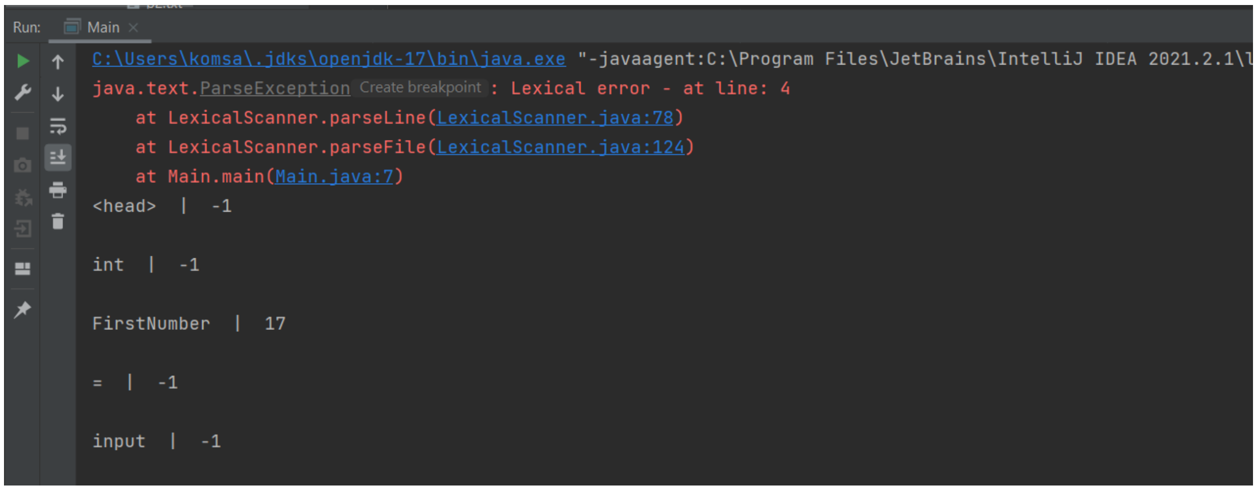
```
;  |  -1

Log  |  -1

(  |  -1

sum  |  1

)  |  -1

;  |  -1

</body>  |  -1

Program is lexically correct!

Process finished with exit code 0
```

p1err.txt:



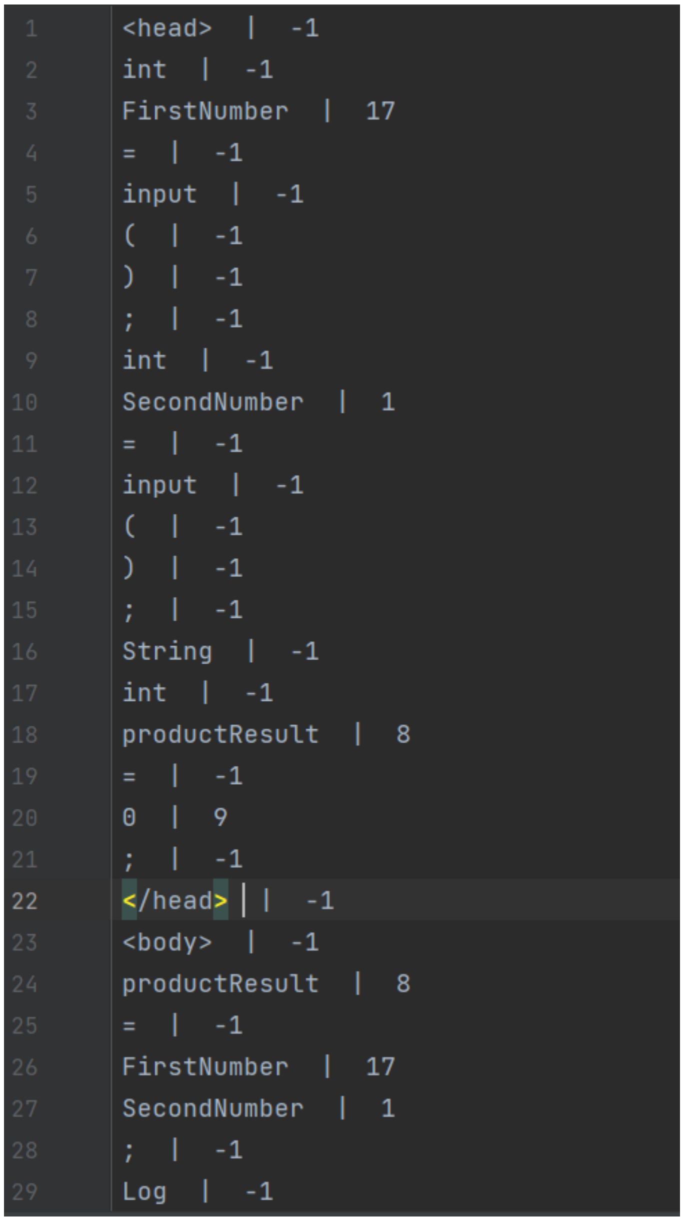
```
<head>
 int FirstNumber = input ();
 int SecondNumber = input ();
 String $Result;
 int productResult = 0;
</head>
<body>
 productResult = FirstNumber times SecondNumber;
 Log (Result, productResult);
</body>
```

Output:



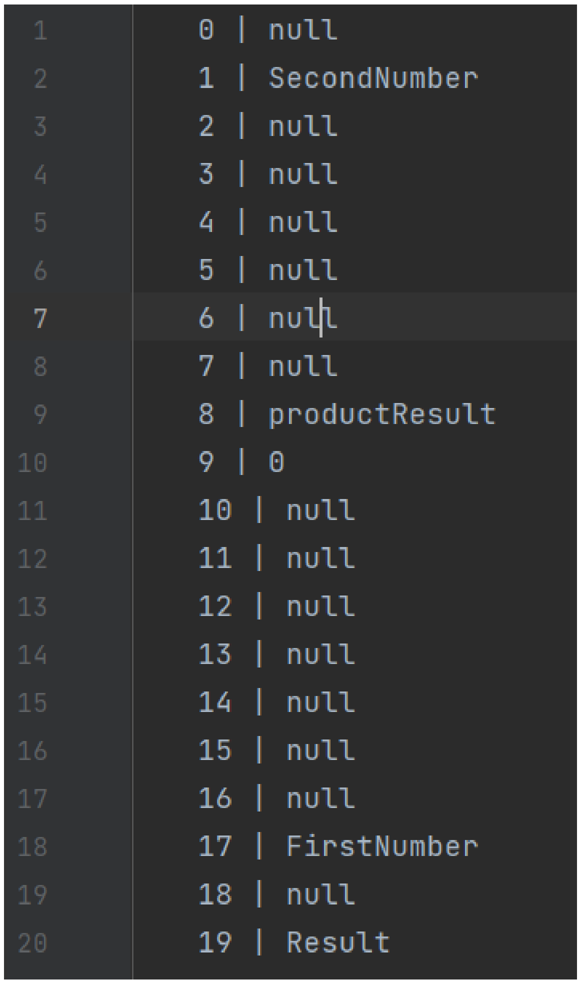
```
C:\Users\komsa\.jdks\openjdk-17\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.1\l
java.text.ParseException Create breakpoint : Lexical error - at line: 4
    at LexicalScanner.parseLine(LexicalScanner.java:78)
    at LexicalScanner.parseFile(LexicalScanner.java:124)
    at Main.main(Main.java:7)
<head>  |  -1

int  |  -1

FirstNumber  |  17

=  |  -1

input  |  -1
```

PIF.out:



```
<head>  |  -1
int  |  -1
FirstNumber  |  17
=  |  -1
input  |  -1
(  |  -1
)  |  -1
;  |  -1
int  |  -1
SecondNumber  |  1
=  |  -1
input  |  -1
(  |  -1
)  |  -1
;  |  -1
String  |  -1
int  |  -1
productResult  |  8
=  |  -1
0  |  9
;  |  -1
</head>  |  -1
<body>  |  -1
productResult  |  8
=  |  -1
FirstNumber  |  17
SecondNumber  |  1
;  |  -1
Log  |  -1
```

ST.out:



```
0 | null
1 | SecondNumber
2 | null
3 | null
4 | null
5 | null
6 | null
7 | null
8 | productResult
9 | 0
10 | null
11 | null
12 | null
13 | null
14 | null
15 | null
16 | null
17 | FirstNumber
18 | null
19 | Result
```