1 Division

The division operator is implemented as described in Chapter 1.5 of [1]. Tests were added (both passing and failing) in order to verify the correct execution of the program, as explained in Section 6.

2 Remainder

The remainder operator is implemented analoguously to the division operator, with the only real difference being that in `Scanner.getNextToken()`, the "%" token does not need to be treated separately, whereas the division symbol could indicate the start of a comment and special care needs to be taken. For this reason, the code in this part was based on how multiplication is handled, whereas other parts are analogous with the division operator of Section 1 and [1]. Tests are described in Section 6.

3 Unary Addition

The unary addition operator is implemented analoguously to the unary negation operator, which was already present in the code. Its main difference from the binary operators described above is that instead of working in the `JBinaryExpression.java` file, changes need to be made to `JUnaryExpression.java`. There is no dedicated JVM instruction for unary addition (contrary to unary negation, which uses `INEG`); for this reason, we use the `NOP` instruction, which does nothing and thus mimics unary addition.

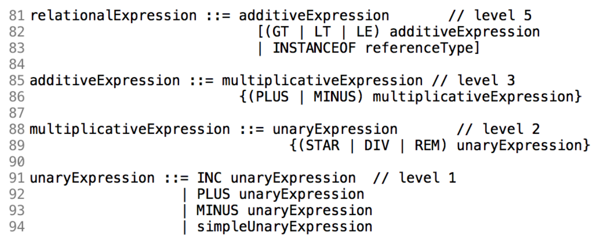
4 Palindrome

To check a string for palindromicity, one traverses the string in both directions simultaneously, continuing as long as both traversed parts are the same. If these substrings differ at some point, the function outputs an empty string and terminates, otherwise it continues until the pointers meet, and outputs the original string. As discussed in [2], this algorithm is optimal from a complexity point of view. To make this algorithm case-insensitive, the string is converted to lowercase before performing the comparison using the built-in Java `String` function `toLowerCase()`.

In order to check whether the middle of the string has been reached, the less-than operator (<) had to be implemented, which was done similarly to the greater-than operator which was already implemented. Tests were also written for this operator, in order to verify its correctness, as explained in Section 6.

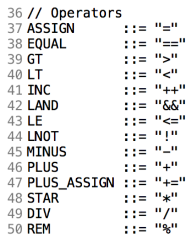
5 Updates to Grammar

The updates to the grammar and lexical grammar are given in Figures 1a and 1b.



```
81 relationalExpression ::= additiveExpression       // level 5
82                           [(GT | LT | LE) additiveExpression
83                           | INSTANCEOF referenceType]
84
85 additiveExpression ::= multiplicativeExpression // level 3
86                         {(PLUS | MINUS) multiplicativeExpression}
87
88 multiplicativeExpression ::= unaryExpression       // level 2
89                                 {(STAR | DIV | REM) unaryExpression}
90
91 unaryExpression ::= INC unaryExpression  // level 1
92                    | PLUS unaryExpression
93                    | MINUS unaryExpression
94                    | simpleUnaryExpression
```

(a) Updated lines: 82, 85, 88, and 92



```
// Operators
ASSIGN      ::= "="
EQUAL       ::= "=="
GT          ::= ">"
LT          ::= "<"
INC         ::= "++"
LAND        ::= "&&"
LE          ::= "<="
LNOT        ::= "!"
MINUS       ::= "-"
PLUS        ::= "+"
PLUS_ASSIGN ::= "+="
STAR        ::= "*"
DIV         ::= "/"
REM         ::= "%"
```

(b) Updated: LT, DIV, and REM

Figure 1: Updates to the grammar (left) and lexical grammar (right) of `j--`

6 Testing Methodology

Tests were added in the same fashion as in [1], i.e., before implementing the functionality, and in the fashion of unit tests. For division and remainder, these tests were really straightforward and make sure the behavior is as expected; all types of scenarios are verified. For unary addition and less-than, adding tests was also fairly straightforward considering the similarities with the operators mentioned before. Finally, to test the palindromicity check, all constraints on the output were checked in order to guarantee correct execution.

The various operators and functions were also verified on INGInious. obtaining a perfect score.

References

[1] Bill Campbell, Swami Iyer, and Bahar Akbal-Delibaş. *Introduction to Compiler Construction in a Java World*. Chapman and Hall/CRC, Boca Raton, Florida, 2012.

[2] Fernando Pelliccioni. *Palindromes and more*. Components Programming, 2016.