Languages and Translators (LINGI2132)
Assignment 1: Report for Writing tests in J –

Group 14: Nihale SABA
Syntyche SHIMBI AMUNASO

February 2016

1 Introduction

Writing tests in j— for new operators involves several steps: Writing pass tests, Encoding JUnit test cases, Adding the JUnit tests cases to the j—test suite and Writing the fail tests.

2 Pass tests

Pass test are programs that are successfully built using the j—compiler. For the division operator, the subtraction assignment operator, the modulus operator and the GCD operator, we wrote respectively, Division.java, MinusAssign.java, Mod.java, GCD.java past tests. These programs are store in the tests/pass directory. The Division.java program includes a divide () method which takes two integer arguments and returns the rounded down quotient of the division between the first argument and the second argument. MinusAssign.java includes a method called minusAssign(int a,int b) which accepts two integers a et b, assigns the difference between a and b to a and returns the value of a. Mod.java includes a method named mod() which accepts two integers as well and returns the remainder of the division between the first argument and the second one. GCD.java contains gcd(int a,int b) which is based on the recursive version of the Euclidean algorithm. This algorithm‘s precondition is that a is larger than b. We wanted our gcd() method to work for case where a is smaller than b. To do so, we added another condition: If (a == 0) return b.

3 Junit test Cases

JUnit Test scenarios execute pass tests and are stored in the test/Junit folder. We wrote DivisionTest.java, ModTest.java, MinusAssignTest.java and GCDTest.java for the division operator, the modulus operator, the substraction assignment operator and for the GCD operator respectively. These test cases extends the junit.framework.TestCase which provides methods such as assertEquals() used to test that two values are equal. Junit also provides methods to setUp() the environment before running the test and method to tearDown() after the invoking the test. Knowing that these methods only accept integers as inputs, we only considered integers value in our test cases.

DivisionTest.java includes testDivide() used to test the divide(int a, int b) method from Division.java. For this, we considered the following cases: a = 0 and b != 0, a = 1 and b ! = 0 ! = 1, a = b, a ! = 0 ! = 1 and b ! = 0 ! = 1.

MinusAssignTest.java contains testMinus() method which evaluates minusAssign(int a, int b) method from the MinusAssign.java. The evaluation involves the scenarios where a ¿ b (a or b are positive or negative integers), a ¡ b (a or b are positive or negative integers), a ¡ b (a and b are both positive or negative integers), a ¿ b (a and b are both positive or negative integers), a = b, a or b = 0.

ModTest.java includes testMod() which tests mod() from Mod.java. The testing includes the following cases: a ¿ b (a or/and b are positive or/and negative), a ¡ b (a or/and b are positive or/and negative), a = 0 and b != 0.

GCDTest.java includes testGcd() method that evaluates gcd(int a, int b) method belonging to GCD.java. As we have already mentioned, we modified the original Euclidean algorithm for GCD found on Wikipedia[1] so that GCD may work for both cases where a ¡ b and b¡a.

4 Test Runner

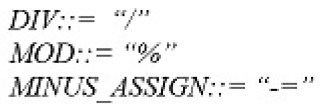
The JMinusMinusTestRunner.java enables the execution of test programs found in test/pass folder. It contains a test suite which groups and executes together several test scenarios. Therefore, we added DivisionTest.class, GCDTest.class, MinusAssignTest.class, ModTest.class to the suite in the JMinusMinusTestRunner.java so that the tests may executed together.

5 Fail test

Fail tests are j– programs that report errors without generating class files for the defective program after compiling. These programs are stored into the tests/fail folder. We wrote Division.java, Mod.java and MinusAssign.java for the division operator, the modulus operator, the subtraction assignment operator respectively. These operators can only function with integers so the compiler should detect the error if operands are of incorrect types. Therefore, in the fail tests, we used char types values (characters) as operands for these operations.

6 Lexical and Syntactic Grammar

We modified the grammar and lexicalgrammar files so that they describe with precision the modified syntax of the language. In the lexicalgrammar file, we added codes depicted in figure 1 in which each line describes respectively, the division, the modulus and the subtraction assignment operations.



DIV::= "/"
MOD::= "%"
MINUS_ASSIGN::= "-="

Figure 1: Added codes to the lexicalGrammar file

In the grammar file, we added the division and the modulus operators' descriptors to the grammar rule describing multiplicative expressions, because they are multiplicative operator. However we included the substraction assignment operator's descriptor to the assignmentExpression.

7 Changes to scanner

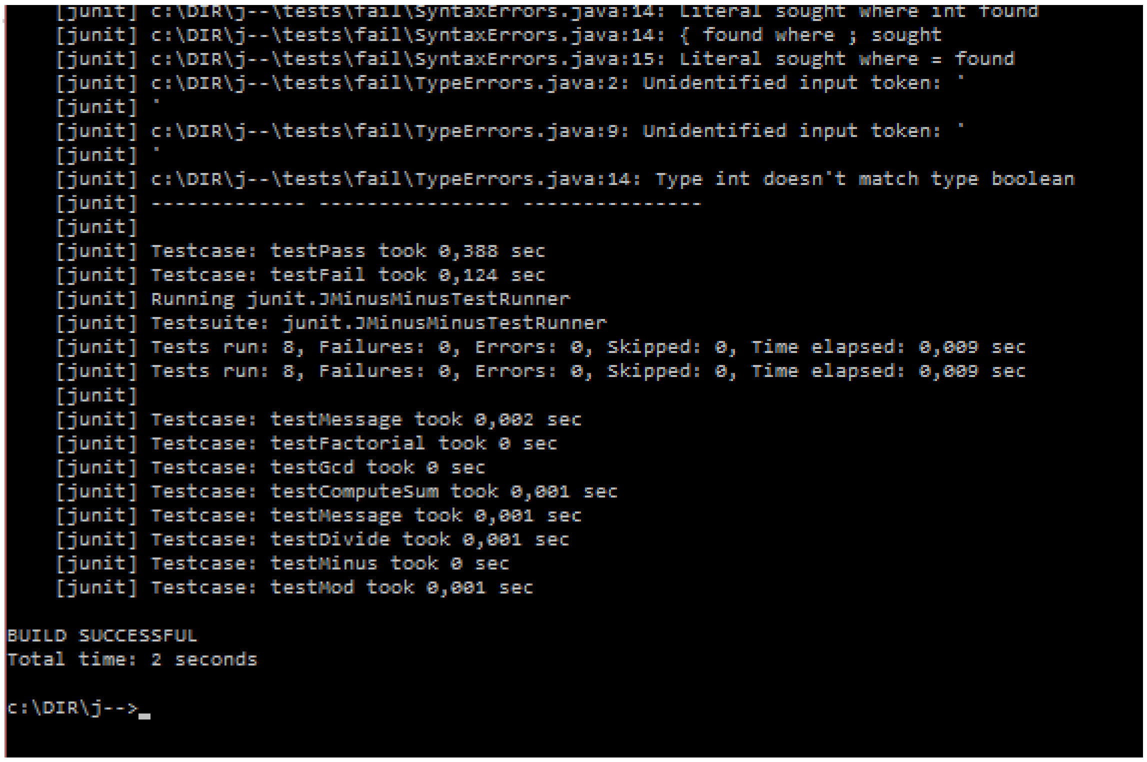
In order to make changes to Scanner.java, we should register the new tokens such as MINUS$_{A}SSIGN, DIV, MOD, byaddingthefollowingintoTokenInfo.javafile.InScanner.javawehavemod$
$asoperators.$

8 Semantic Analysis and Code Generation

In the file JBinaryExpression.java belonging to src/jminusminus folder, we implemented JDivideOp class for the division operator and JModOp class for the modulus operator. Each of these classes contains a constructor, analyse() and codegen() methods. In JAssignment.java we implemented JMinusAssignOp class for the substraction assignment operator . Each constructor in the above classes creates respectively, an AST for a division, modulus expression given its line number, its left and right operands. Analyse() evaluates the operands of the corresponding operations, ensuring that each type is integer and setting the produced expression's type to int. codegen() generates code for the two operands involved, and emits JVM (IDIV) instruction for the (integer) division of two numbers inside the JDivideOp class. However it emits JVM (IREM) instruction for the (integer) division of two numbers inside the JModOp class. In JMinusAssignOp class, codegen() emits ISUB for the subtraction assignment.

9 Test output

The program was successfully built and the results are the expected ones. The figure below depicts the results of ours test programs in command line.



```
    [junit] c:\DIR\j--\tests\fail\SyntaxErrors.java:14: Literal sought where int found
    [junit] c:\DIR\j--\tests\fail\SyntaxErrors.java:14: { found where ; sought
    [junit] c:\DIR\j--\tests\fail\SyntaxErrors.java:15: Literal sought where = found
    [junit] c:\DIR\j--\tests\fail\TypeErrors.java:2: Unidentified input token: '
    [junit] '
    [junit] c:\DIR\j--\tests\fail\TypeErrors.java:9: Unidentified input token: '
    [junit] '
    [junit] c:\DIR\j--\tests\fail\TypeErrors.java:14: Type int doesn't match type boolean
    [junit] ------------- ---------------- ---------------
    [junit]
    [junit] Testcase: testPass took 0,388 sec
    [junit] Testcase: testFail took 0,124 sec
    [junit] Running junit.JMinusMinusTestRunner
    [junit] Testsuite: junit.JMinusMinusTestRunner
    [junit] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0,009 sec
    [junit] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0,009 sec
    [junit]
    [junit] Testcase: testMessage took 0,002 sec
    [junit] Testcase: testFactorial took 0 sec
    [junit] Testcase: testGcd took 0 sec
    [junit] Testcase: testComputeSum took 0,001 sec
    [junit] Testcase: testMessage took 0,001 sec
    [junit] Testcase: testDivide took 0,001 sec
    [junit] Testcase: testMinus took 0 sec
    [junit] Testcase: testMod took 0,001 sec

BUILD SUCCESSFUL
Total time: 2 seconds

c:\DIR\j-->_
```

Figure 2: The output of the tests execution

References

[1] Wikipedia. *Euclidean algorithm.* 2016 (accessed on February 5, 2016). Accessed from: http://books.google.com/books?id=W-xMPgAACAAJ.