LINGI2132 - Assignment 1

The purpose of this assignment is to become familiar with Eclipse, the j-- code tree and the overall structure of the compiler. You will have to touch every phase of the compiler: lexical analysis, the parser, semantic analysis and code generation. But all of your code should follow the pattern of code already present in the j-- compiler, so you should not have too much trouble.

JUnit Tests

For this assignment (and the others), you will have to write JUnit tests for your implementation. Some examples are already provided in the eclipse project. The correction of your implementation will be based on your tests. Moreover, we will use other tests (to which you do not have access), so do not neglect this part.

Static Code Analysis

Source code style is too often neglected by programmers. We will pay attention to this in your implementation too. To do so, we will use CheckStyle (http://checkstyle.sourceforge.net/), a static code analysis tool for Java. Especially, we will use eclipse-cs (http://eclipse-cs.sourceforge.net/), a plug-in that integrates Checkstyle into Eclipse. If your coding style is bad, your grade will be affected negatively (until 20% of the maximum grade).

To be able to use eclipse-cs in Eclipse, follow the instructions on http://eclipse-cs.sourceforge.net/basic_setup_project.html .

You should create your own CheckStyle configuration (http://eclipse-cs.sourceforge.net/basic_creating_config.html).

For this assignment, the following modules will be used to check your code :

- Size violations
- Coding problems
- Duplicates
- Metrics
- Whitespace

Also, you may use File Sets to check only your files (http://eclipse-cs.sourceforge.net/advanced_file_sets.html) .

Submission

To submit your work, we ask you to export your Eclipse project as a **zip** file (right click on the project → export... → General → Archive File). You **have to** use the following name convention for your zip file : j--_gr_*grnumber*.zip , where *grnumber* is your group number on iCampus. If you do not follow this convention, the script will not pass and your work will not be graded.

You have to submit this zip file on iCampus (in Assignment/Assignment_1(code)).

Submit also a **brief** report (pdf format) on iCampus (in Assignment/Assignment_1(report)) that explains what you did. 1 page is sufficient, go straight to the point.

Submit in time, late submissions are not possible and therefore will not be graded !

Correction Script

We will use a bash script to correct this assignment. It will be provided to you on iCampus (Documents) so you will be able to test your work before submitting. This will help prevent unpleasant surprises. To test it, simply pass your zip file as an argument. Do not forget to give execution right to the script.

Exercises

First, make sure you have the j-- tree, http://www.cs.umb.edu/j--/j--.zip on your computer.

1. Unzip j--.zip in a directory of your choice (let us call it *DIR*). The path to DIR **cannot** have white spaces or you will have problems at some point (also it is a bad practice). You now should have a directory called **j--** in DIR. Open Eclipse and switch workspace to DIR. Create a Java Project from the existing Ant buildfile found in DIR/j--. To do so, in Eclipse, click File → New Project … . Select "Java Project from Existing Ant Buildfile". Select the "build.xml" file in the unzipped j-- directory. **Call your project j--**. Click Finish. That's it ! Your project is set up. You can check if the set up is correct by making a right click on the build.xml file and selecting Run as → Ant Build. You should see BUILD SUCCESSFUL at the end of the console.
2. Read through Chapter 1 carefully, and study the code in the code tree. This will take time.
3. Modify the example compiler to add the division operator / as described in Chapter 1.
4. Modify the compiler to add the modulo % operator.
5. Modify the compiler to add the unary plus + operator.
6. Do exercise 1-12. Use JUnit tests to test your j-- program. We add the following conventions : the class must be placed in the test/pass package, must be named "Primes" and have a non-static "primes" method with the following signature : public int[] primes (int n). This method must return the prime numbers less than or equal to n, in increasing order and without repetition. Those conventions **must** be respected for the tests to pass !
7. Write a narrative describing how you went about writing and testing each program you write. The narrative should describe what choices you faced and which you made. We should get a pretty good sense of what your program looks like from reading the narrative. So far as testing goes, describe how you went about choosing your test cases, what they were, and their results.

While seemingly easy, this assignment requires a certain amount of setup and so can take some time; don’t put this off! **The due date is firm**.

Deadline: 14/02/2014, 11.59 pm.

Grades

Narrative (/5)

- Clear discussion of what you did, didn't do, and why.

- Clear discussion of development: choices available to you, choices taken, and why.

- Clear discussion of testing: coverage, tests chosen, test results.

Programs and Testing (/15)

For this part, the grades will be binary : either your programs pass our tests (and of course yours too!), either they do not. So it is in your interest to write as many **useful** tests as possible. An implementation is good only if it is correct in any case.