CS2112—Spring 2012

Homework 4

Parsing and Fault Injection

Due: Thursday, March 15, 11:59PM

When people build compilers and bug detection systems, it is painful and expensive to hand-code programs that contain errors. A common approach to generating such programs is to use *fault injection* to transform existing, presumably correct programs into incorrect programs that presumably should be detected as such by the compiler or bug detector. A fault injector works by making some number of random, local edits to the correct program. These are the faults that are being injected.

In this assignment, you will build a parser for a simple language. You will build a *pretty-printer* that can print out parsed programs. And you will build a fault injector that changes programs into other programs.

This assignment is also the first part of the group project for the course. The programming language you will be parsing, pretty-printing, and injecting faults into will be the language controlling simulated creatures. You will need to read the project specification to find out more about the final project and the language you will be working with in this assignment. The faults that are being injected are those corresponding to the mutations in Section 9 of the project specification.

0 Changes

- (March 5) We originally asked you to implement the `Token` and `Tokenizer` classes. To save you time, we have now decided to provide these implementations to you.
- (March 6) We are now requiring that you submit an early draft of your design overview document. Also, the nature of the faults to inject was clarified.
- (March 7) Clarified the requirements for pretty-printing.

1 Working in a group

You will work in a group of two students for this assignment. Obviously, it is important that you find your partner very soon.

As usual, we encourage you to read all Piazza posts because often your questions have already been asked by someone else—even before it occurs to you to ask. You will save a lot of time this way.

Working with a partner may be challenging. Some tips:

- Meet with your partner as early as possible to work out the design and to divide up the responsibilities for the assignment. Keep meeting and talking as the project progresses. Be

prepared for your meetings. Be ready to present proposals to your partner for what to do, and to explain the work you have done.

- This project is a great opportunity to try out *pair programming*, in which you program in a pilot/copilot mode. It's fun. It also tends to result in fewer bugs. A key ingredient is to give the person typing the job of convincing the other person that the code meets the spec. Of course, you need to agree on a spec first.
- It might seem as if you are working harder than your partner. But remember that when you go to implement something, it typically turns out to be twice as hard as you thought. So what your partner is doing is also twice as hard as it looks. If you think you are working twice as hard as your partner, you're probably about even!
- We encourage you to use a version control system such as Subversion or Git to manage your code so that you and your partner can work separately when necessary.

2 Overview document

Starting with this assignment, we expect your group to submit an overview document. You will want to read the Overview Document Specification to learn what we are expecting. Writing a clear document with good use of language is important. We are also asking that you submit an early draft of this document.

3 Parsing

Your program should be able to parse any legal critter program according to the grammar given in the project specification.

Optional: If given an illegal program, it should report a syntax error and give the line number of the input file on which the error occurred.

4 Pretty-printing

The pretty-printing functionality should be able to print out programs in the same syntax that it was written in. So the output of your program should be readable by your program. It is expected that the output will use indentation and line breaks to make it readable while fitting within an 80-column line width. You should not use ASCII tab characters in the output.

5 User interface

Your program must support the following command-line interface:

- `java -jar <your_jar> <input_file>`
  parse the file `input_file` as a critter program and pretty-print the the program to standard output.

- `java -jar <your_jar> --mutate <n> <input_file>`
  parse the file `input_file` as a critter program and apply n mutations. After each mutation, print a description of the kind of mutation that has been applied, and pretty-print the program.

6 Programming tasks

You will want to figure out with your partner how to break up the work involved in this assignment. To get you started thinking about this, here are some of the major tasks involved:

- Implementing the main program and command-line handling.
- Designing and implementing a class hierarchy of classes for representing abstract syntax trees. These will be subclasses of `Node`. We have given you a start on some of these classes, but you will likely need to add more.
- Implementing the `Parser` class to generate abstract syntax trees.
- Implementing pretty-printing functionality, as methods on AST nodes.
- Implementing a class or classes to perform fault injection. It is up to you to design the interfaces for these classes.

Note that we originally asked you to implement the `Token` and `Tokenizer` classes. We have decided to provide these to you.

7 Restrictions

You may use any standard Java libraries from the Java SDK. However, you may not use a parser generator.

8 Overview Draft

We are requiring you to submit an early draft of your design overview document on Monday before the assignment is due. You may not be able to predict what your design and testing strategy will look like in full at that point, but we want to see how far you have gotten. We will aim to get you quick feedback on this draft.

9 Submission

You should compress exactly these files into a zip file that you will then submit on CMS:

- *Source code*: You should include all source code required to compile and run the project.
- *Tests*: You should include code for all your test cases.
- `overview.txt/html/pdf`: This file should contain your overview document.

Do not include any files ending in `.class`. We expect you to stick to Java 6 features and avoid features found only in Java 7.