Deliverable
As the outcome of this project, the students will form groups for the remaining projects. Each group will write a short report and send it to the TA. The report should list the members of the new groups. The report should provide the following information for each group member:
• Language: In which language was the scanner/parser implemented.
• Public Test Score: Report on the percentage of passed/failed public tests on the project's due day. In addition, provide a brief description (several sentences) on what was the most challenging part of the implementation.
• Hidden Test Score: Report on the percentage of passed/failed hidden tests on the project's due day. In addition, provide a brief description (several sentences) on what was the most challenging part of the implementation.
• Comparison With Previous Results: Report how much the percentage of passed tests (both public and hidden) has improved since the testing before obtaining the hidden tests.

In addition, the group should write a one paragraph plan on how they intend to integrate their scanners/parsers in the next project (and give a rationale for their decision).

6.035 Group Formation Report
arkadiyf, jamb, mayars
"Random Compilers"

jamb:
Language: Java
Public Test Score: 3 failed tests for scanner (out of 28), 0 failed tests for parser (out of 61) = 96.6%. Figuring out how ANTLR worked for the scanner was very challenging and after a while I gave up with a mostly working scanner and moved on to the parser. I found the parser much easier since it was mostly just figuring out how to phrase the rules given in the decaf spec and no unintuitive formatting. I was also frequently confused until I discovered that I needed to touch the Main.java file between each run of ant or else it would stop working for some reason.
Hidden Test Score: 0 failed tests for scanner, 0 failed tests for parser = 100%. Basically the only modification I needed to make once hidden tests were released was recognizing that "\n" is an acceptable character but an actual newline in the middle of the file isn't. This wasn't challenging per se but I was confused about it and had to ask Maya to clarify.
Comparison: 5 additional scanner tests passed after I made the modification above. The rest were already working. All of the parser hidden tests passed on the first try, and the parser public tests were working by the time the hidden ones came out.

mayars:
Language: Java
Public Test Score: passed all scanner + parser tests. I had a lot of trouble setting up Git and finding appropriate things to read on ANTLR. Once I found the version 2 documentation, things went more smoothly. One of the biggest challenges was calling newline() when there was a

linebreak after a single quote, primarily because locating the exception syntax in ANTLR took a few hours. On the parser, I sometimes mistyped things, e.g. omitting a semicolon, misspelling a token name, or missing one part of a complicated expression. I solved this by whittling down the file to the minimum broken example and using --debug.
Hidden Test Score: passed all scanner + parser tests. I made no changes; the most challenging parts were getting the test scripts to run on my machine (the testall.sh referred to scanner/test.sh instead of tests/scanner/test.sh for instance, and some of the bash commands didn't exist on my machine)
Comparison: I made no changes after pulling the hidden tests; everything passed on my first try.

arkadiyf:
Language: Scala
Public Test Score: All of the scanner tests and parser tests passed. The most challenging part of the work was getting all of the operations to scan correctly, even going so far as to make parser test legal-19 fail because of a scanning issue. When trying to group operations, I got many nondeterminism errors, and when things seemed to work, it felt more like a quirk of ANTLR than me solving the problem. I finally solved everything by separating them into completely separate tokens for most of them. Additionally, I had some difficult recognizing different characters, but this went away instantly once I realized I could enter ASCII ranges and they would get interpreted correctly. When working on the parser, there was some small amount of difficulty with getting expressions to parse correctly, but that was quickly resolved by starting with just multiplication and addition and then working up from there. It was also hard to use the debug feature, as even in simple examples I found myself getting lost in the wall of text provided. Besides that, most of the work was straightforward from copying the provided Decaf spec.
Hidden Test Score: My code passed all of the parser and scanner tests without any added modifications. As such, I did not have any extra difficulties after dealing with the public tests, and so this portion of the project did not take me any extra time.
Comparison:I made no changes after pulling the hidden tests; everything passed on my first try. While I finished my work after the release of the hidden tests, I specifically did not run them until I was confident with my work on the public tests.

Integration for subsequent project parts:
Since Maya and Jackie both wrote in Java, we'll use Maya's perfectly-working scanner and Jackie's perfectly-working parser as a base (with some token names changed to match). Each group member will look them over and make suggestions for changing things if they think their version did a given thing better. In particular, Arkadiy plans to integrate a lot of his parser ideas.