CPSC 326: Homework 4 *Due: Mar. 19*

Goals:

- Think about and write code to performan semantic analysis (checking).
- More practice using the Visitor pattern by implementing visitor functions to "semantically check" ASTs (programs).
- Continue practicing working with unit tests, creating tests, etc.

Instructions:

1. Use the GitHub Classroom link (posted in Piazza) to copy the starter code into your own repository. Clone the repository in the directory where you will be working on the assignment.
2. Copy all of the files from your HW3 `src/main/java/cpsc326` directory, *except* for `MyPL.java`, into your HW4 `src/main/java/cpsc326` directory.
3. Complete the `SemanticChecker` visitor implementation `SemanticChecker.java`. To start, you should stub out the visit functions (so that the code can compile).
4. Ensure your code passes the unit tests provided in `SemanticCheckerTests.java`. (Note you will want to do steps 3 and 4 iteratively.)
5. Ensure your code works with the example files within the `examples` subdirectory. Note that the examples should all "check out", i.e., no semantic errors should be found.
6. Create additional unit tests as specified in the TODO comment at the end of `SemanticCheckerTests.java`.
7. Create a short write up as a **pdf file** named `hw4-writeup.pdf`. For this assignment, your write up should provide a short description of the unit tests you created and any challenges and/or issues you faced in finishing the assignment and how you addressed them. The description of the tests can be short, but should state why you designed the tests the way you did (i.e., justify why the test is non-trivial / interesting, and what it is actually testing).
8. Submit your program by ensuring all of your code and writeup is pushed to your GitHub repo. You can verify that your work has been submitted via the GitHub page for your repo.

Additional Requirements: Note that in addition to items listed below, details will also be discussed in class and in lecture notes.

1. Additional tips and tricks for this assignment will be discussed in class. You will likely have a number of questions—be sure to post them early to piazza to give enough time for them to be answered and for you to understand the answers. You should also be monitoring piazza for the questions being asked (and the answers).
2. You can not deviate from the general visitor approach and functions specified in the starter code. Similarly, you may not modify any of the AST classes provided.

3. You will need to allow yourself a good amount of time to think through some of the trickier parts of the semantic checker (there are many!). If you start this assignment too close to the deadline you will likely run out of time.

4. If you use any print statements for debugging, you must remove these from your final solution. In addition, you must remove all commented out code from your final submission.

Homework Submission and Grading. Your homework will be graded using the files you have pushed to your GitHub repository. Thus, you must ensure that all of the files needed to compile and run your code have been successfully pushed to your GitHub repo for the assignment. Note that this also includes your homework writeup. This homework assignment is worth a total of 40 points. The points will be allocated according to the following.

1. **Correct and Complete (30 points).** Your homework will be evaluated using a variety of different tests (for most assignments, via unit tests as well as test runs using specific input files). Each failed test will result in a loss of 4 points. If 6 or more tests fail, but some tests pass, 6 points (out of the 30) will be awarded as partial credit. Note that all 30 points may be deducted if your code does not run, large portions of work are missing or incomplete (e.g., stubbed out), and/or the specified techniques, design, or instructions were not followed. Because assignments build on each other, in most cases you will need all tests to pass before moving to the next assignment.

2. **Evidence and Quality of Testing (5 points).** For each assignment, you must provide additional tests that you used to ensure your program works correctly. Note that for most assignments, a specific set of tests will be requested. A score of 0 is given if no additional tests are provided, 1–4 points if the tests are only partially completed (e.g., missing tests) or the tests provided are of low quality, and 5 if the minimum number of tests are provided and are of sufficient quality.

3. **Clean Code (2 points).** In this class, "clean code" refers to consistent and proper code formatting (indentation, white space, new lines), use of appropriate comments throughout the code, no debugging output, no commented out code, meaningful variable names and helper functions (if allowed), and overall well-organized, efficient, and straightforward code that uses standard coding techniques. A score of 0 is given if there are major issues, 1 if there are minor issues, and 2 if the "cleanliness" of the code submitted is satisfactory for the assignment.

4. **Writeup (3 points).** Each assignment will require you to provide a small writeup addressing challenges you faced and how you addressed them as well as an explanation of the tests you developed. Additional items may also be requested depending on the assignment. Homework writeups do not need to be long, and instead, should be clear and concise. A score of 0 is given if no writeup is provided, 1 if parts are missing, and 2 if the writeup is satisfactory.