CPSC 326: Homework Assignment 3 *Due: Tuesday, Feb. 14*

Goals:

- Implement a basic recursive descent parser for MyPL (to check syntax but not build the AST);
- Practice writing and working with unit tests.

You are free to use whatever IDE and machine you prefer for this class. However, to complete the assignment, you will need `git`, `g++` (version 11 or higher), `cmake`, `make`, the google test framework, and `valgrind` installed. It can also be useful to have a debugger such as `gdb` installed as well. Each of these are already installed on the remote development server (`ada.gonzaga.edu`) provided by the CS Department. However, you may also install these programs on your own machine, via a virtual machine, running WSL2, or on your own remote server. Note that you will also need a GitHub account for obtaining starter code and for submitting your assignment.

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 (e.g., onto `ada` or your own machine).
2. Modify your `mypl.cpp` program from HW-2 (see below).
3. Copy your lexer implementation (`lexer.h` and `lexer.cpp`) from HW-2 into the `src` directory for HW-3 along with your `mypl.cpp` file. Your lexer is required to implement the parser.
4. Complete `SimpleParser` by adding and implementing recursive descent functions based on the MyPL grammar provided in Lecture 5.
5. Ensure your code passes the unit tests provided in `simple_parser_tests.cpp` within the `tests` subdirectory.
6. Ensure your parser correctly handles the example file `parser.mypl` within the `examples` subdirectory. Note that nothing should be output by your implementation for the file (including no errors detected).
7. Create five additional unit tests as specified in `simple_parser_tests.cpp`. Include a description of the tests you created in your homework writeup.
8. Create a short write up as a **pdf file** named `hw3-writeup.pdf`. For this assignment, your write up should provide a short description of any challenges and/or issues you faced in finishing the assignment and how you addressed them along with your new unit tests. Be sure your `hw3-writeup.pdf` file is in the main directory of your assignment (and **not** within the `src` directory or any other subdirectory).
9. Submit your program. Be sure to add, commit, and push all assignment files 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, in lecture notes, and in the discussion section.

1. Modify your `mypl.cpp` file from HW-2 so that the `--parse` flag calls your parser (supporting both standard input and file input). Your code for calling the simple parser should look something like the following. Note that the snippet below assumes that the lexer has already been created.

```
try {
  SimpleParser parser(lexer);
  parser.parse();
} catch (MyPLException& ex) {
  cerr << ex.what() << endl;
}
```

2. While you should generally follow the MyPL grammar when defining your recursive descent functions (i.e., one function per non-terminal), you will likely find that in a few places it is easier and/or makes sense to combine grammar rules into a single recursive descent function and define a few additional helper functions.

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 30 points. The points will be allocated according to the following.

1. **Correct and Complete (24 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 files). Each failed test will result in a loss of 2 points. If 10 or more tests fail, but some tests pass, 4 points (out of the 24) will be awarded as partial credit. Note that all 24 points may be deducted if your code does not compile, large portions of work are missing or incomplete (e.g., stubbed out), and/or the specified techniques, design, or instructions were not followed.

2. **Evidence and Quality of Testing (2 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 if the tests are only partially completed (e.g., missing tests) or the tests provided are of low quality, and 2 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 (including file headers), 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. In addition, when compiled, your code should not have any warnings. 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 (2 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. 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.