CPSC 326: Homework Assignment 7 *Due: Tuesday, Apr. 18*

Goals:

- Develop a better understanding of VM-based stack machines and their use in PL interpretation.
- Implement a code generator for MyPL that takes an AST and translates it to corresponding VM instructions to be run by the interpreter.
- Finish the basic `mypl` program implementation.
- Write additional test cases ("interesting" programs) using MyPL and run them on your finished implementation.

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. Finish the `CodeGenerator` implementation by completing the `visit` functions in the `code_generator.cpp` file.
3. Update your `mypl.cpp` file to support all options, including `--ir` to print the VM instructions generated (see below).
4. Ensure your code passes the unit tests provided in `code_generator_tests.cpp` within the `tests` subdirectory.
5. Run the example programs given in the `examples` subdirectory and make sure your implementation works correctly over each.
6. Create **three** new and "interesting" MyPL programs and execute them using your finished MyPL implementation. Interesting programs do something useful and go beyond the provided programs in terms of exercising the capabilities of MyPL. Note that you will be graded on the quality of the programs you come up with as part of testing. Name your programs `prog1.mypl`, `prog2.mypl`, `prog3.mypl` and add then to the main directory of your assignment. (Note that the most involved program a student has written so far was a a chess simulator with ascii graphics!)

7. Create a short write up as a **pdf file** named `hw7-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 evidence (screenshots) that your `hw7` program is working correctly. The screenshots should show the running of your `mypl` impelementation on the test files within the `examples` directory as well as on your three interesting programs. Briefly state in the write up what you chose as your three programs and why. Be sure your `hw7-writeup.pdf` file is in the main directory of your assignment (and **not** within the `src` directory or any other subdirectory).

8. 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 Information. You must update your `mypl.cpp` file to support the `--ir` flag and to run the code generator and VM in "normal" mode. Your `--ir` flag should look something like this:

```
try {
  ASTParser parser(lexer);
  Program p = parser.parse();
  SemanticChecker t;
  p.accept(t);
  VM vm;
  CodeGenerator g(vm);
  p.accept(g);
  cout << to_string(vm) << endl;
} catch (MyPLException& ex) {
  cerr << ex.what() << endl;
}
```

The "normal" mode should look something like this:

```
try {
  ASTParser parser(lexer);
  Program p = parser.parse();
  SemanticChecker t;
  p.accept(t);
  VM vm;
  CodeGenerator g(vm);
  p.accept(g);
  vm.run();
} catch (MyPLException& ex) {
  cerr << ex.what() << endl;
}
```

Finally, to support both the built-in string and array `length()` functions, you will need to update your semantic checker to distinguish the names of the two functions. In particular, in my implementation, the semantic checker changes the name of the array-based "`length`" function to "`length@array`". In the code generator, function calls to "`length`" generate `SLEN()` instructions whereas function calls to `length@array` generate `ALEN()` 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.