Compiler Design
Project 2
Expressions with variables

1 Extending the language for variables

To add variables to our programs that comprise a list of statements (each of which is an expression), we prefix the statement list with a declaration list:

```
[x, y, _quux13]
(= x 5)
(= y 13)
(= _quux13 (- y x))
```

Notice both the allowable form of the variable names as well as the addition of an assignment operator.

2 Getting started

Our source code will now be on *GitLab*, into which you can login with your college credentials. Be sure, too, to set up *ssh keys*. (If you set them up in a previous semester, they should still work; if you haven't see this ssh key setup help page.)

You can use your browser to see the Project-2 repository, and to grab it so that you can work with it, clone it like this…

```
$ git clone git@gitlab.com:amherst-college/academics/compilers-2526F/project-2.git
```

This respository contains some initial files:

- `Compiler.java`: Modified from Project-1 to include a call to a *verifier* that performs *semantic analysis* through a call to a method named `bind()`. This method will walk the AST, connecting declared variables to their uses.
- `Lexer.java`: A new lexer that scans our new elements, namely *square brackets* (`[]`) and *names* (e.g., `foo_quux_23`).
- `Token.java`: Now includes tokens for the new elements mentioned above.
- *My solution to 1-C*: The rest of the source code, as well as the `grammar.txt` file, is from my completed Project 1-C code.

In this form, the code will not compile and run. However, it is a starting point that includes a working lexer, as well as parser and generator that can handle *KapLang, version 1.* From there, you can modify the code to handle variables.

3 Your assignment

Our three groups should modify this code to augment the parser, implement the verifier, and modify the generator so that expressions that use variables (including assignment expressions) work properly.

Using `git`: Each group should *create its own branch*, modifying and adding code to perform that group's given task. A group can create special *tester classes* that are used to debug and refine their group's code. This tester code should be clearly named (e.g., `VerifierTester.java`) so that it does not interfere with any other code. We will later merge all of these branches, including the handling of any conflicts, as a class exercise when the assignment is due.

4 How to submit your work

Since this is a large group project, submitting the group's work will simply entail a *commit/push* of your work in the repository.

This assignment is due on Friday, Oct-10, 11:59 pm.