Interpreter/Compiler Assignment #1: Expressions and Assignments

Issued: Tuesday, February 14
Due: Thursday, March 2

Purpose

This assignment asks you to extend a provided translator, for a small programming language. The translator is both an interpreter and a compiler. The translator is implemented in Java. Its source language was invented for this assignment. Its compiler's target language is C.

Source Code and Grammar

An implementation of the translator is at:

```
pub/ia1
```

The translator employs an ad-hoc scanner and a recursive-descent parser. The parser builds a strongly typed parse tree, which is then traversed and processed. A grammar for the source language is:

```
stmt  : assn ';'
assn  : id '=' expr
expr  : term addop expr
      | term
term  : fact mulop term
      | fact
fact  : id
      | num
      | '(' expr ')'
addop : '+'
      | '-'
mulop : '*'
      | '/'
```

You can compile and run the translator, with commands like these:

```
javac *.java
java Main "x = 1+2;" "y = x+3;"
```

Note that you must quote Bash metacharacters on the command line. A regression tester, described below, provides a better way to run the translator. Note further that this runs the translator on two complete source programs, according to the grammar.

The example, above, only runs the interpreter, not the compiler. To run both, generating a C source file, set a Bash environment variable to the desired file name. For example, these commands generate a C file named `gen.c`, produce an executable file, and execute it.

```
Code=gen java Main "x = 1+2;"
gcc -o gen gen.c
./gen
```

The translator's design is based on the object-oriented design patterns named Interpreter(243) and Builder(105), from the well-known "Gang of Four" textbook used in CS 472.

Assignment

There are several parts:

- Test the translator thoroughly. I have provided a simple regression tester, named `run`. Add tests to my rudimentary test suite. IF YOU DO NOT USE THE REGRESSION TESTER, YOUR SUBMISSION WILL NOT BE GRADED.
- Finish documenting the provided source code. For example, see method `past` in `Scanner.java`.
- Extend the scanner to support source-code comments. Design your own form of comment.
- The grammar specifies that a source program is exactly one assignment statement. However, `main` translates each of its command-line arguments as a separate source program, trying to modify variable values, in the `Environment` object, accordingly. Currently, this is broken. Change the interpreter to fix it. Don't bother changing the compiler to fix it. Eventually, we will extend the grammar, thereby obsoleting this "feature."
- Add a prefix unary minus operator (e.g., `-(x+3)`). This is a change to the grammar and parser. Simply changing the scanner to allow negative numbers is insufficient.
- The translator currently supports only integer values. Change it to *instead* support double values. Of course, an integer can be represented as a double.