Translator Assignment

Issued: Thursday, September 28
Due: Thursday, November 2

There are two parts to this homework assignment. The first part allows you to become familiar with an existing simple translator, by documenting its source code and making small changes to it. The second part allows you to enhance your translator, to interpret and compile a much more realistic programming language.

You are *strongly* encouraged to ignore Part 2, until you have fully completed Part 1. Please reread the previous sentence.

The provided translator is:

```
pub/ta
```

It is implemented in Java, its source language was invented for this assignment, and its compiler's target language is C.

Part 1: Expressions and Assignments

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:

```
pub/ta/Grammar1
```

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 example 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 also 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 provided translator thoroughly. I have also 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.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 feature 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.

Part 2: Statements

If you have not completely finished Part 1, please stop reading Part 2, and return to Part 1.

Move your Part 1 solution into a subdirectory, named `part1`. Copy your `part1` subdirectory, creating a directory at the same level, named `part2`. Perform Part 2 of this assignment in your `part2` directory. When you are all done, make one submission, from the parent directory of `part1` and `part2`.

As before, your 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 extended source language is:

```
pub/ta/Grammar2
```

Assignment

There are several parts:

- Extend your scanner to recognize the new keywords and operators.
- Extend your parser to recognize the new statements and expressions.
- Extend your evaluator/generator to translate the new constructs.
  - You can represent boolean values as double values (e.g., 1.0 and 0.0);
  - To interpret I/O statements, read from `System.in` (hint: use a JDK `Scanner`) and write to `System.out`.
  - To compile I/O statements, read with `scanf` and write with `printf`.
- Test your solution thoroughly. Add tests to your test suite. The quality of your suite will influence your grade.