CSE 6341, Programming project 3

Due Monday, October 12, 11:59 pm (30 points)

The goal of this project is to build an interpreter for the language from Project 2 (with some minor changes to the language, as defined below). The semantics to be implemented was discussed in class when we considered the definition of operational semantics. Your implementation will use the code for AST building and type checking from Project 2, with some minor changes to `Interpreter.java`. In your interpreter, first perform the type checking from Project 2 and exit with error code `EXIT_STATIC_CHECKING_ERROR` if the checking fails. If the checking succeeds, execute the program.

Set up

Copy your implementation from Project 2 into `.../proj/p3` and do `make clean`. Then edit `Interpreter.java` to add the following new exit codes:

```
public static final int EXIT_UNINITIALIZED_VAR_ERROR = 3;
public static final int EXIT_DIV_BY_ZERO_ERROR = 4;
public static final int EXIT_FAILED_STDIN_READ = 5;
```

Restriction

To simplify the project, the following restriction will be imposed on all input programs: no two variables have the same name. This also applies to variables that are in different blocks: so, code such as **int z = ...; { ... { int z = ...; } }** is never going to be part of a valid input program. You do **not** have to check that this restriction is satisfied by the input program: just assume that it is and implement your interpreter under this assumption.

Details

The semantics was described in class. A few additional details:

1) You have to catch run-time errors for "use of unintialized variable" and "division by zero" and then exit the interpreter with the corresponding error codes.

2) **readint** and **readfloat** expressions should read from UNIX *stdin* and produce a value of the corresponding type. If this reading cannot be performed successfully, the interpreter should exit with error code `EXIT_FAILED_STDIN_READ`. A simple way to implement this functionality is to use a `Scanner` object from `java.util`: e.g., at the very beginning of program execution create an object `Scanner s = new Scanner(System.in)` and then call `s.nextLong()` and `s.nextDouble()` whenever you need to evaluate a **readint** or **readfloat** expression. When executing the interpreter from the command line, you can put the input data in some file `datafile` and then do `./plan programfile < datafile`

3) Type INT in our language should be implemented by a Java `long` type; type FLOAT in our language should be implemented by a Java `double` type.

4) Since each variable declaration uses a unique name (due to the restriction described above), you can implement the state **σ** with one map from variable names to values. There is no need to use a tree of maps. At the beginning of execution, the map is empty.

5) The evaluation of boolean **&&** and **||** operators should use short-circuit semantics.

6) A natural way to implement the checking is to add to each expression class a method `evaluate` which takes as input a reference to the state, evaluates the expression, and returns the resulting value. Similarly, for each statement class you can use a method `execute` which takes as input a reference to the state, executes the statement, and as a result changes the state. However, feel free to ignore this suggestion completely.

Testing
Write many test cases and test with them. Submit at least 5 test cases with your submission. The test cases will not affect your score, but I will examine them to see how you approached testing. Put them in the same location as the provided file t1 and name them t2, …

Submission
After completing your project, do
```
cd p3
make clean
cd ..
tar -cvzf p3.tar.gz p3
```

Then submit `p3.tar.gz` in Carmen.

General rules (copied from the course syllabus)

1) Your submissions must be submitted electronically via Carmen by midnight on the due date. The projects must compile and run on **stdlinux**. Some students prefer to implement the projects on a different machine, and then port them to stdlinux. If you decide to use a different machine, it is entirely your responsibility to make the code compile and run correctly on stdlinux before the deadline. In the past many students have tried to port to stdlinux too close to the deadline, leading to last-minute problems and missed deadlines.

2) Projects should be done independently. General high-level discussion of projects with other students in the class is allowed, but you must do all design, programming, testing, and debugging independently. Projects that show excessive similarities will be taken as evidence of cheating and dealt with accordingly. Code plagiarism tools may be used to detect cheating. See more details in the Syllabus under "Academic integrity":
http://web.cse.ohio-state.edu/~rountev.1/6341/pdf/CSE6341-Syllabus-Au20.pdf

3) The projects are due by 11:59 pm on the due day. No exceptions will be made to this deadline: if you submit at 12:00 am, your submission will be late. Please plan your time carefully and do not submit in the last minute. You can submit up to 24 hours after the deadline; if you do so, your project score will be reduced by 10%. If you submit more than 24 hours after the deadline, the submission will not be accepted, and you will receive zero points for this project.