CPS710 Project: VNM Interpreter

Part IV: Evaluation

Preamble

In this part of the project, you will write an evaluator that visits the nodes of the syntax trees produced in Part 3.

Preparation

Part 4 of the project handles the final stage of the interpreter, evaluation. You may use your solution to part 3 as a starting point, or you may use the provided model solution.

Create a working directory for this part of the project – e.g. ~/cps710/project/part4/ or similar. Copy the files from the following directory into this folder:

```
cp -r /usr/courses/cps710/aufkes/project/part4/code/* .
```

For this part of the project, you can work from your own solution, or work from the provided model solution. See the setup below for each option.

Use your own solution:

Copy your VNM.jjt into the working directory and add the following option:

```
VISITOR_EXCEPTION="Exception";
```

Use the makefile under **code/use_your_own/** to generate VNMEval.java.

Use provided solution: (VNM.jjt not provided until after part 3 is due)

Find the VNM.jjt and VNMEval.java under **code/use_provided/**. Copy these into your working directory.

Additionally, in the **src/** directory, you will find an updated TestVNM file, as well as the various token classes. Use these token classes or use your own. They will need to be in your working directory as well before you compile everything.

From here, run make (or make -B if it insists things are already “up to date”) to recompile everything using the makefile in the **code/** directory. Make sure you also have the token classes, and TestVNM.java in your working directory. If everything compiles as it should, you’ll have the AST directory created once again with all the AST nodes present.

What to do:

In this part of the project, you will implement an evaluator for a handful of AST nodes from the VNM language. You won't implement evaluators for everything, and you can assume at this point that the VNM sentences being entered are error-free.

If you look inside the VNMEval.java file in the "use_provided" folder, you will see that visit methods for ASTprint and ASTstring are already filled in. This means you can try executing the "run" script and typing print statements with strings as arguments. For example, the statement below should echo back "helloworld".

```
print "hello", "world";
```

If you generated your own VNMEval.java, you can use the one that was provided as a reference point for your own ASTprint and ASTstring visit methods. You should write visit() methods for the following nodes. The first two should be fairly easy, given ASTprint and ASTstring as starting points.

- String literals, numeric literals, and Boolean literals
- Print and println, which should work with Boolean, integer, or string arguments.
- Integer arithmetic, including addition, subtraction, multiplication, and division. These operations should work the same way they do in Java (including precedence and associativity). For example, "**-3+5-10-2;**" should evaluate to -10.
- Comparison: This can be a Boolean literal, or a comparator operation between numeric literals. E.g. **3<=6**.
- If statements: An if statement can be driven by a condition, or by a Boolean literal. In an if statement, only the first true clause should evaluate. For example, when the following if statement evaluates, it should print "Hello":

```
if #1 then
  println "Hello";
fi;
```

Similarly, the if statement below should print "World"

```
if 2>5 then
  println "Hello";
elif #1 then
  println "World";
else
  println "abc123";
fi;
```

Testing & Marking

Testing is just like the previous parts. You can execute the 'run' script to enter sentences manually, or you can use the 'runtests' command to evaluate the provided test cases.

To run all tests, first compile your code (run make) execute the "runtests" script (after giving yourself execute permissions). As always, you can edit the runtests file and comment out/in individual tests as needed.

Like previous parts, your mark will be based on the proportion of tests that you pass.

Submission

Zip up your VNM.jjt and VNMEval.java files into an archive called vnm_part4.zip. If you created any additional Java files, such as helper classes, be sure to include those in your zip file as well. Submit on moon using **"submit-cps710aufkes vnm_part4.zip"**