Name: Sean Rettig
Date: 2015-03-15

Milestone 5 Report

Specification (what do <u>you</u> think the purpose of this milestone is)

The purpose of this milestone was to continue learning the process of designing and implementing translators. In this milestone, we built on our gforth generator that we created in the previous milestone to work with loops and variables.

Processing (how did <u>you</u> go about solving the problem)

In order to do variable declarations, type checking, type conversions, etc., I basically had to redo all of Milestone 4 since I did not originally write it in a way that would allow for these features. It was also fairly buggy and hacky; the new version, while still not perfect, works much better.

My lexer would originally not handle tokenizing type declarations well, which I didn't notice until this milestone since we hadn't been using variables. The lexer modification also caused some bugs in my parser, but those were fairly easy to fix.

My syntax tree generator was not working properly in situations that involved let statements or extra parentheses (such as those used for scoping), and I ended it rewriting it about 4 times before I finally found a fairly elegant method that seemed to work well. Part of the reason for my issues with the tree generation was also that I also had to track down a bug in my tree implementation where every node would automatically inherit the children of all previous nodes (due using an empty list as a default arg in Python).

I also ended up needing to design another data structure for my symbols instead of trying to use my token data structure for everything.

Name: Sean Rettig
Date: 2015-03-15

Once I got all that out of the way, I realized that I should have started sooner on this project, since I have run out of time and was not able to finish it. I did start on implementing scope by using a scope stack that is searched through in reverse when a variable name is used, and I laid the groundwork for variables to be used, I just ran out of time before I could implement the gforth translation code that does the CREATE stuff. I also wasn't able to finish loops. However, automatic conversions of types seems to work quite well; adding ints produces ints, adding reals produces reals (using f+), and adding an int and a real performs a 's>f' automatically to make it work. Using addition with strings also automatically subs in “s+”, as well as converting the strings to gforth format first (same with reals; an “e” is appended to reals if one was not already present). All of my passing tests produce valid Gforth.

Testing Requirement (how did <u>you</u> test for correctness)

I have 12 test files which are intended to pass, i.e. produce valid gforth code. Each file tests different features, such as integer and real arithmetic, automatic int to real stack moving, comparisons, nested statements, multiple statements, trigonometric functions, boolean logic, and string concatenation. I feel that these tests are decently comprehensive. They all pass! I also have a couple of non-passing tests that use wrong arguments.

Retrospective (what did <u>you</u> learn in this milestone)

This milestone required personal research and a TON of trial and error to get this stuff working. It would have been really nice if we had spent a few weeks going over the

Name: Sean Rettig
Date: 2015-03-15

processes needed for this milestone in class rather than all of the different types of parsing that we ended up not even using.