David Silva

Phase 3 Write-Up

Problem Motivation

During phase 3, the objective is to generate code based on our lex and bison files from the previous two phases. To do this, we had to use our lex/bison files from previous phases and modify them to generate code. Mainly the bison file needs modification.

Approach

My approach mainly consists of using one class to “bubble up” code chunks and “return values.” I have a CodeBlock class that has the main following features: store intermediate code chunks, store lists of variables, store labels, and store label locations, store “return” values.

The reason I put return in quotes is because I know that the rules do not “return”, but instead it is the value I want the parent rule to use for its own code generation.

Each parent rule also has access to the code chunk that “bubbles up.” This is mainly for loops and applying labels correctly; appending or prepending labels to the appropriate code chunks/blocks. If I have a continue statement I generate a label, generate a “goto” label statement in the code, and store the label generated within the class so that I can print it right before the boolean expression to determine whether or not to run the loop again. The previous sentence explains the reasoning behind storing labels. The reason I store label locations is for error checking. If I have a “dangling” label that did not get placed before a boolean expression, I know that the continue statement is not within a loop. So I must print out the label line/location to give proper error messages.

Storing lists of variables is for read/write functionality. So that I can read/write one variable per line of code generated. I have no way of passing to my sequence of

variables rule whether a read or write is the parent rule, so I must store the list of variables and generate the proper code in the read/write rules themselves.

Lastly, I have a list of globals that I use: a symbol table, a list of functions, a list of code generated. I also have a temp variable generator and label generator.

Part Each Student Did

N/A

Specific Problems Encountered

The only real problem I encountered was placing labels properly. I had most of the project done and the last part of code generation I was working on were do while loops. Other loops evaluated a boolean expression first, so I was able to just append label statements within the boolean expression itself. However, a do while always runs once and I must prepend a label to the entire loop. I had no way of doing this without using a class. So I refactored my entire project to use a class. Properly interfacing the class with bison was the hardest part from then on. I designed the class in such a way that it would work with my already placed code. I also designed the class to keep error messages in mind.