Checkpoint 3 Report

Lance Paje, Pasan Undugodage

Construction Process

During the construction of the compiler, we tried to follow the slides on chapter 11, TMSimulator, especially the incremental steps towards the end of the slides. This gave us the basis on how we were supposed to approach this checkpoint. We first created a separate file called CodeGenerator.java to do our work for Milestone 3. This file contained its own set of visit methods much like the previous Milestones had with SemanticAnalyzer.java and ShowTreeVisitor.java. We had to make a different accept function in each of the absyn files to match each of the visit methods used in CodeGenerator.java. These visit methods contained an integer offset and a Boolean describing if what was being passed was an address alongside with a reference to the class found in the absyn folder. The visit functions in ShowTreeAddress and SemanticAnalyzer were also modified in such a way to allow for calls in CodeGenerator to go off.

The CodeGenerator.java was then modified to include some set values such as address sizes and register values at the top of the file for later use in processing the c minus file we sent in. Specifically these variables were used for help in processing the emit routines later down the line. Afterwards various emit routines were created to help with scribing what's going on in the file itself. These emit routines would be used by the visit functions to describe what is happening to the code. Some visit functions, such as trees, also print to file using the emit routines.

We modified the CM.java file to accommodate for the -c operator according to the checkpoint description so that it prints out the contents of a newly made .tm file. We did not reach the point of properly writing down the contents of a file however due to a bug that skips on main as soon as it reaches it so nothing is printed.

Overall Design

The overall design closely follows what was laid out in the slides, specifically chapter 11, TMSimulator. We used a CodeGenerator file to contain most of our code generation code. We modified the visit method to contain an offset and a boolean isAddress. This new visit method replaced the old visit method. The visitor model is still used heavily for this Milestone. We added an int funaddr in FunctionDec to contain the address of a function. We added a nestLevel and offset on the VarDec abstract class so that it is inherited by the SimpleDec and ArrayDec. This is used for keeping track of their global level and stackframe memory address respectively. We created emit routines for processing code generation, and copied the ones in the slides. emitLoc points to the location of the current instruction being generated and highEmitLoc points to the next available space for printing. We made it as far as the 4th incremental development slide and could not continue due to a bug that did not present us with the anticipated output.

Assumptions and Limitations

We made assumptions that the files passed into our CodeGenerator would be.cm files and not other file types and as such did not account for any errors that could arise should a different file type is passed through the CodeGenerator. Another assumption made was that even if an error is made, the semantic analyzer or parser would catch it first. In essence we trust our previous work to catch errors beforehand

and either perform a recovery or stop the process entirely. As a result a limitation of this program would be in its lack of error checking or error recovery. For instance, if there was an array, we would always assume it would be of type int because otherwise it would be caught by the semantic analyzer.

We considered ourselves limited by our lack of knowledge towards Assembly and were not primed well enough to tackle this checkpoint. Many headaches were had in trying to trace and retrace the program. We were also limited in our efforts to debug our own code due to the interconnected nature of this project at such a low level setting. Given that it's low level it's easier for errors to pop up since there is no pre existing garbage cleanup for memory aside from what we write ourselves.

Possible improvements

For one more descriptive variable names could be used in our code. At first we considered using short variable names as we have methods that use many variables in their parameters such as emitComment() in order to keep the length of each line relatively short. However this came at a trade off of having less descriptive variable names and in turn it was easy to forget what each variable did and in turn harder to debug our code. It didn't help that the same variables were being used in different spots of the code which made deciphering what we intended to use them for much harder. Another solution to this would have been to add comments that describe each variable better with a lengthier name.

Another improvement that could have been made was time management. One of our team members had difficulty in participating in this checkpoint in a meaningful way as they had projects due at the same time as this one, even with the extensions. More

time was also found trying to figure out how TMSimulator worked and it ended up not helping much in the long run in comparison to the length of time spent on it. Perhaps simply reading more of the slides would have helped more efficiently.

Creating more helper functions would also help in the long run, especially helper functions focused on debugging. This could be done through an automated test case model provided by Java.

Issues Crossed

At first we had problems figuring out the Assembly code as it had been a while since we touched Assembly and so it was a huge hindrance to our progress until we could get ourselves how to write Assembly code again.

We had trouble with error handling this time around again. Specifically we had trouble getting the correct values to appear for the pc variable in CodeGenerator. We were unable to figure out why the value was incorrect, despite trying to trace it manually through the file using print statements.

This problem extends from having to see multiple parts of the code work at once before anything can be seen working, given that we had to get the helper functions working properly first before trying to do anything else. At times we would have to go around fixing other things before we could see any improvements.We figure that the error with the pc variable could be caused by a different set of problems as the pc variable is used everywhere in the code.

The fact that the visit function is also being called in other files such as SemanticAnalyzer.java and ShowTreeVisitor.java was something that we didn't look into and could further obfuscate the problem. It's entirely possible that previous non-issues

in the previous Milestones could also crop up as issues being manifested as incorrect outputs.