CIS*4650 W24 Checkpoint 3 Project Report

Salman Azhar (1058981)

Overview:

In this report I will first be going over the problems I faced for this checkpoint, which hindered me from completing it. First, I will be going over the technical issues and then the conceptual difficulties. I will also be discussing what I would have done differently In order to complete it.

Related techniques: Inheritance, visitor pattern, hashtables and function overloading.

Issues faced:

Technical issues:

- The primary technical issue I faced in the process of starting this checkpoint was the task of implementing the -c flag
- The issue was regarding how to ensure that the syntactic analysis would be performed before moving on to the code generation.
- The syntactic analysis and code generation should be happening in the SemanticAnalyzer.java file.

- The problem I was having with this was not being able to separate the -s flag and -c flag functionality. Integrating the code generation into the semantic analyzer would require the code to somehow know if it is being run as a syntactic tool or a code generation tool. I will discuss the possible solutions to this in the “Possible solutions” sections.

Conceptual issues:

- The conceptual issues were the main hindrance in the process of completing this checkpoint.
- Although I understood the idea of a static runtime environment in a broad sense, I could not quite keep up with the minute intricacies of the implementation.
- I struggled with the various pointers, offsets and locations that were needed to keep track of. I understood what they were tracking, but didn't quite understand the role they play in the actual processing and compilation of the program.
- I also had trouble understanding some of the emit functions. Particularly the backup, skip and restore functions.

Possible solutions:

Technical issue Solutions:

- For the technical problems, a possible solution could have been to have a property such as “gen_code” and “gen_table” which would be passed to the SemanticAnalyzer. These properties would determine if the assembly code and symbol table are generated, respectively.
- To accomplish the -c flag, the semantic analyzer would be called with the gen_code flag set to true and gen_table flag set to false. This would result in all the symbol table print functionality being disabled and the emit functions printing abilities enabled.

Conceptual Issue Solutions:

- The solution to the conceptual problems I faced would have been mostly solved by starting the checkpoint earlier and also attending office horse with the questions I had.
- Additionally, I could have followed a more incremental approach to the checkpoint by just starting off with implementing the setup first (I/O, Prelude and Finale).

Reflection:

Assumptions, limitations and possible improvements:

The only assumption in my code is that the grading for the syntax and semantic analysis will come from running the -a and -s flags. As for limitations, the code

still handles all the c1 and c2 requirements, but is missing the c3 requirements. Possible improvements include starting earlier on the checkpoint and diligently seeking help in office hours for parts where I was confused. Also, attending lectures more often and going through the textbook as well. Also, I could have improved the implementation of how my code prints to the output files, although that works perfectly fine for c1 and c2, simplifying it could have greatly helped with getting started on c3.