Project Report: Checkpoint 3

Group Members:

- Eason Liang - 1146154
- Thomas Phan - 1017980
- Raeein Bagheri - 1149021

1. What was done

1.1 Overall Design and Implementation Process

This Checkpoint involved a couple of things which included us needing to refactor a couple of stuff written in checkpoint one and checkpoint two needing to add extra flags and also needing to add codeGenerator.java which includes us needing to generate assembly code. With this current checkpoint and the implementation process out group was given a lot more resources with slideshow 11 which provided clear understanding on the subtasks and the resources and information to do checkpoint three. The first thing our team looked to understand was the assembly code which was provided by the professor. The one that was provided was the TMSimulator which provided a TM.c and also a list of the test files originally provided in checkpoint one. At the same time our team looked at the subtasks which involved some refactoring that was done with checkpoints one and two which included a couple of flags and also new attributes that were needed to be added in the abstract syntax tree. After that, our group looked at Subtask three which just involves us generating the correct code with a simple void main(void) {}. Otherwise, our group looked to implement the correct assembly instructions by adding to each visit function. Finally, our group looked to cleanup was trying to match the output that the professor provided in TM Simulator Packages which proved to be a very difficult task in itself.

1.2 Lessons Learned

Throughout the project, we encountered several technical and logistical challenges, including:

- **TM Simulator and Assembly:** In this checkpoint, we learned the main concepts and language provided in assembly. In this case, it was totally different from standard assembly but also completely different from the Easy68k Simulator. So one thing needed to learn was the language which our group found to be the most difficult part in this checkpoint. Another thing is understanding how the tm .c that was provided works and how it essentially runs the code as that's needed in order to run the code. Another thing we needed to learn was the actual simulator itself which was different from courses like CIS*2030 although somewhat similar as they provided features such as tracing, step by step for each instruction and also running the entire program there were a lot of things that we found hard and learned if we looked to better ourselves when it came to time management.
- **CodeGenerator class:** In this checkpoint, we were required to implement a code generator class which required us to implement the Emit function to generate

assembly code for each C-Minus file. This required us to understand the TM simulator package provided by the professor which again required us to understand how to run the simulator and also see how the assembly code should be generated.

- **Error checking:** This required us to understand and also clean up the Abstract syntax tree, and semantic analysis in order to generate the correct errors. As well we also needed to consider runtime errors which we found it hard to implement which in this case for was checking if the index was out of bounds.

1.3 Significant Modifications

When looking at the checkpoint and any significant modifications that were made throughout the checkpoint only a couple were made changes which followed what the professor has provided us. One was needing to add new flags and attributes so in this case it mentioned needing to add the isAddr flag in each of the visit functions and also needing to add funaddr and boolean flag to it. So when looking back at checkpoint one and checkpoint two in this case there weren't many significant changes that were needed. In this case, in checkpoint two the indent was a bit off for symbol two and something we want to look into fixing.

1.3 Possible and Future Improvements

When looking at the possible improvements that could be done through the checkpoint there was a lot due to the time crunch. We found it rather difficult to implement checkpoint three and understand the checkpoint which in this case was mostly focused on understanding the assembly code. After that, the implementation was another difficult hurdle that needed to be done which involved getting the correct instructions but also in this case getting the correct addresses and values as well. I think a couple of improvements that could be done are going and fully implementing it correctly getting the correct addresses and correct instructions being printed out for the checkpoint three code generation. Furthermore, another thing we can look at is code optimization leading us to run code faster and also generate more optimal code where we have fewer instructions in the assembly code. Finally one thing is making the error checking overall more robust as there were specific tests that did not do error checking correctly. Another thing we wanna look to do in the future is checking for runtime errors so any errors such as the index out of bounds error was something that was not implemented in this checkpoint three.

1.4 Assumptions

In this checkpoint 3 due to the work needed to be done there wasn't much assumption that needed to be considered since a lot of the outline and guidelines were provided to our group both in the description and in the slide show which did help when it came to the assumptions as well. Furthermore we were also provided with the test files and the correct output which again reduces any vague instructions or questions that may have come along the way.

Furthermore We always assumed that Raaein is always at the gym 🙂 and also we need to Assume that the code works and is fully functional (It doesn't)

2. Contributions of each member

In our collaborative effort to meet the objectives of checkpoint three for our compiler project, we divided the workload to leverage each team member's strengths and interests, ensuring that contributions were equally distributed among us. Collectively, we all engaged in creating the documentation and adding the grammar rules necessary for parsing the C- language, creating CodeGenerator.c to generate the correct assembly code, and also looking to document and test all the programs. To highlight this each group member

Members	**Contributions**
Eason Liang	- Implemented starting CodeGenerator.java file - Implement Prelude and Finale and also input and output for the assembly code - Implement EM functions - Implement isAddr flag and added boolean flags - Adjust the visit function to ensure these work with the new flags and attributes - Implement starting code for Void main()

	- Create Test Cases - Documentation - Add -c flag for code generation - Implement boolExp
Raeein Bagheri	- Create Test cases - Helped with creating ArrayDec - Helped with matching the assembly code with samples given - Updated makefile - Provided emotional support where needed
Thomas Phan	● Implemented type-checking for Assignment Expressions ● Implemented type checking for Function declaration and argument passing ● Implemented Operator type-checking ● Support my teammates in their implementations ● Created test cases ● Implemented if, while, SimpleVar visit functionality ● Helped to implement AssignExp visit class