Project Report: Checkpoint 2

Group Members:

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

1. What was done

1.1 Overall Design and Implementation Process

The implementation process for checkpoint two differed significantly from checkpoint one. Unlike checkpoint one, we did not receive any starting files. Consequently, our best reference point for implementation was Professor Song's lecture notes. These notes provided detailed guidance on how to proceed with implementing checkpoint two, focusing on tasks such as traversing the tree to create the semantic analyzer, establishing the symbol table, and implementing type checking. To begin, we revisited tree traversal, similar to our approach in checkpoint one. However, this time, our objective was to not only traverse the tree but also to construct the symbol table and output the scope. To achieve this, we utilized a hashmap where each scope served as a key, paired with an ArrayList containing the values within that scope. After our team was confident with the output of the symbol table, our team moved into implementing type and error checking within checkpoint two which was long and tedious.

1.2 Lessons Learned

During the implementation of checkpoint one, our team encountered several learning opportunities that significantly impacted our approach to the project and future incoming milestones. The key lessons we learned include:

- **Communication:** A factor and lessons learned that greatly facilitated our team's progress was effective communication. Collaborating and openly discussing implementation details on checkpoint two allowed us to understand the requirements of the checkpoint and exchange different ideas. Despite fewer meetings during checkpoint two compared to checkpoint one, I found that our team was way consistently active.
- **Pair Programming**: In checkpoint one, because many multiple components could be divided within multiple teammates which made it more efficient. However, during checkpoint two, splitting up tasks for different components proved a challenge due to the cohesive nature of the work, mostly with the focus being on the symbol table. Recognizing the importance of shared understanding and collaboration, we looked at pair programming as an alternative approach. This decision not only made understanding the checkpoint easier but also made it relate to communication and why it was a key thing.
- **Symbol Table, Error Checking, Type Checking**: While concepts such as symbol tables, error checking, and type checking were covered in class, translating this theoretical knowledge into practical code presented considerable

difficulty. However, when implementing these concepts in checkpoint two made each team member solidify their understanding of building a semantic analyzer. These tasks include building a symbol table, understanding the scope, and implementing type checking, therefore helping our team's understanding of these fundamental concepts.

-

1.3 Assumptions

A couple of assumptions we operated on, especially with generating the symbol table, is whether we generate it through the files or the standard output. In this case we looked to generate the semantic analysis output in the file when running with the -s flag. Furthermore, we made the assumption that user input from the test cases always included a main at the end of the *.cm* file.

1.3 Challenges

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

- **Symbol Table**: Implementing the symbol was the first thing that needed to be implemented rather confusing at the beginning needing to utilize a hashmap to store the scope required a lot more understanding than anticipated since we needed to traverse the tree and also understand the CM- language to understand where a new scope is being declared. Also in terms of debugging it becomes

difficult to debug especially when there's no correct output to base our results off so there were many times that our team got lost due to this.

- **Type Checking**: Implementing Type Checking was rather difficult since there are many cases needing to be checked. Such examples can be checking whether typing checks if it's int, bool, or void. In this case, there are many more that need to be checked to validate their type checking such as function definitions and function prototypes but also include any computations and any assignment statements. Finally the most difficult is checking within the if and while blocks checking whether an if block is an actual boolean condition. Overall this left this to be a very long and tedious process.

2. Contributions of each member

In our collaborative effort to meet the objectives of checkpoint two 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. Additionally, each member focused on:

Members	**Contributions**
Eason Liang	- Created test cases - Implemented the error reporting functions

	- Implemented the Hashmap for storing the symbols - Create the documentation for checkpoint two
Raeein Bagheri	- Added the skeleton files to start the project - Helped with creating the test cases - Implemented type checking for If Expressions - Implemented type checking for function calls - Implemented type checking for function return types - Implemented type checking for variable declarations
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