Project Report: Checkpoint 1

Group Members:

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

1. What was done

1.1 Overall Design and Implementation Process

The team decided to approach the compiler construction for checkpoint one in multiple stages and used the SampleParser provided as a starting point. We started with the development of a scanner for the C- language using Jflex which is responsible for tokenizing the input source code. Following that, the focus shifted to implementing a parser that incorporates the grammar rules of the C- language and connects to the scanner using CUP. Next, we added the embedded code within these grammar rules to facilitate the generation of syntax trees, which are crucial for the subsequent compilation phases. Finally, we incorporated error recovery mechanisms to handle syntax errors gracefully and be able to report multiple errors in each run of the parser.

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:

- **Effective Use of Git**: We discovered the importance of using Git for version control effectively. This involved regularly committing our work to keep track of changes, using branches to experiment with new features without affecting the main codebase, and resolving merge conflicts when they occurred.
- **Clear Communication**: Our project benefited greatly from clear and consistent communication among team members. We established regular meeting schedules and used communication tools to share updates, ask questions, and provide feedback. This helped in ensuring everyone was aligned with the project goals and current tasks, reducing misunderstandings and duplications of effort.
- **Starting Early**: One critical lesson was the value of starting the assignment early. By beginning our work well ahead of deadlines, we allowed ourselves enough time to thoroughly understand the project requirements, plan our approach, and divide tasks among team members effectively. This also provided us with a buffer to address unexpected challenges without rushing, leading to a higher quality of work in the end.
- **Step-by-Step Approach**: Initially, we attempted to work on multiple components of the compiler—such as the parser, scanner, and error recovery—simultaneously. We learned that this approach was inefficient and often led to confusion and overlapping work. By refocusing our strategy to complete the checkpoint step by step, we were able to concentrate our efforts on one component at a time, which improved our productivity and the overall quality of each part.

1.3 Challenges

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

- **Configuring CUP**: Setting up CUP on each team member's development machine presented initial hurdles. Differences in operating systems and development environments required us to invest time in creating a standardized Makefile and a readme file with a setup guide and FAQ for common issues.
- **Grammar Ambiguity and Precedence Rules**: Adding precedence rules to CUP to address grammar ambiguity was more complex than anticipated. It required a deep understanding of the parsing process and the specific syntax of the C- language to ensure that our grammar was accurate.
- **Error Recovery**: Implementing robust error recovery mechanisms for syntactical and lexical errors proved to be a significant challenge. Crafting a parser that could gracefully handle and recover from a wide array of potential errors in test files demanded thorough testing and a comprehensive understanding of possible error scenarios.

1.4 Assumptions, Limitations and Constraints

During the initial phase of our compiler development for the C- language, we operated under the assumption that the input programs would largely conform to the expected syntax, featuring only occasional and relatively simple syntactic or lexical errors. This assumption was to enable us to concentrate our efforts on laying a robust foundation for the compiler without the immediate need to tackle complex error handling. However, this approach also introduced a notable limitation in our project's current capability, particularly in the area of error recovery. At present, our compiler lacks the sophistication to address complex syntactical and lexical errors effectively. The incorporation of advanced error recovery strategies, capable of gracefully handling a broader spectrum of errors, would undeniably elevate the complexity of both our scanner and parser components. Considering the primary goal of achieving a functional compiler that aligns with the checkpoint one requirements, we decided to postpone the development of more elaborate error recovery mechanisms.

2. Contributions of each member

In our collaborative effort to meet the objectives of checkpoint one 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	● Jflex implementation ● Implement Production Rules ● Implement Code AST Java Classes ● Implement Error Recovery
Raeein Bagheri	● Added to the Grammar rules

	● Added to the implementation of ShowVisitorTree class ● Created the report file ● Added to the AST Java Classes
Thomas Phan	● Grammar rule CUP specifications and action rules ● Generation of Abstract Node for AST ● Create functions for visiting nodes in ShowVisitorTree class ● Performed error recovery during parsing process