Checkpoint 2 - Project Report

Team Members: Brendan Moorehead, Ali Asgarian

Checkpoint 2 of the compiler project introduces semantic analysis through the implementation of a symbol table and type checking. While Checkpoint 1 focused on lexical and syntactical analysis to generate an abstract syntax tree (AST), Checkpoint 2 extends this by detecting and reporting semantic errors. This is achieved by traversing the AST post-order using a new `SemanticAnalyzer` class. Additionally, the compiler now supports the `-s` flag to output the symbol table. This document outlines our design choices, challenges faced, and lessons learned.

Functionality Implemented:

The symbol table is implemented as a hash map of array lists where each entry represents a scope, and each scope contains symbols (variables, functions) defined within it.

Type checking ensures the correctness of expressions and assignments based on predefined C- types (`INT`, `BOOL`, `VOID`).

Design Choices and Implementation:

We implemented a post-order traversal strategy in the `SemanticAnalyzer` class to analyze expressions only after their sub-expressions have been resolved. This ensures that type checking is done bottom up, dependencies are resolved before usage, and symbol table entries are complete before being referenced.

We designed the compiler to recover gracefully from errors rather than terminating at the first issue. This allows multiple errors to be reported in a single execution.

Implementation Challenges:

Ensuring symbols were correctly accessible across different scopes was challenging. We resolved this by maintaining a global scope lookup while prioritizing local scope resolution. The symbol table originally did not track function redeclarations. We added checks to compare function signatures and report conflicts.

Lessons Learned:

1. Incremental development is key, and early symbol table implementation made debugging easier.
2. Structuring error messages made debugging much simpler.

Assumptions:

- Variables must be declared before use.
- Functions must have explicit return types.
- VOID type cannot be assigned to variables.

Team Contributions:

As a pair, we worked on a symbol table implementation and nested scope management. We also updated Main.java to handle -a and -s flags and wrote the test cases. Ali worked on error handling and reporting, and developed semantic analysis logic for expressions and functions. Brendan developed the AST traversal logic and visitor pattern, and implemented type checking mechanisms.