CIS*4650 Project:
Milestone 2

Ben Carlson 1044277 carlsonb@uoguelph.ca

Conor Roberts 1056167 crober35@uoguelph.ca

March 2022

Summary (TLDR)

This checkpoint/milestone involved the creation of a symbol table and performing semantic analysis on all c minus code. The primary deliverables for this milestone are as follows:

1. Create a package containing relevant symbol classes (ie: array, variable, function)
2. Create a symbol table, a data structure used to store all the symbols in our program's structure
3. Create robust error recovery, that reports semantic errors and can recover

Most of this milestone focused specifically on building a data structure and traversal methods. However towards the very end of the milestone, we decided to integrate our semantic analyzer into our CUP file, in order to make things flow more effectively.

Features of Semantic Analysis

1. Declaration of variables out of scope
2. Type checking
3. Function verification
4. Expression validation, (function calls, etc)

Techniques utilized
When creating the program, we attempted to follow best practices of OOP, abiding to SOLID. Each stage of the development process was done incrementally clearing compiler errors and warnings as necessary. After that we switched into full debug mode with our test files to ensure we had quality product for demo day. Our development process followed 4 stages with one final debug at the end.

Basic Understanding/Design
For this assignment, we first reviewed lecture materials regarding syntax trees and semantical analysis. By doing this we could get a surface level grasp of the implementation details. As we continued down the road of this, we had to review concepts such as postorder traversal, as we were both rusty in this area.

SymbolTable.java
After building the symbols class package, we split apart into two, focusing on both the semantical analyzer and symbol table separately. For the symbol table we opted to use a stack of hashmaps, for ease of access/use. Many lookup, create and delete methods were created in order to seamlessly integrate with the semantic analyzer class.

SemanticAnalyzer.java
Since the symbol table was a partial requirement for this class, we worked specifically on the visitor methods that would not need immediate integration. This served well, until some of the main driving methods relied on the symbol table. Ultimately we finished the table a little later than expected and both ended up sprinting through the visitor methods to ensure we could have enough test time. Visitor methods took a very similar format to that of the first milestone, removing a select few methods that would not be required here.

Debugging
Once these elements were complete, we pair programmed the remaining aspects of the project. By doing this we could quickly be on the same page for what was required to be done in terms of testing and create effective test files for the project.

Learning Experiences

- Learned about different modules of a compiler and their interaction
- Learned about semantic analysis and how it serves as a "sanity check"

for programmers

Assumptions
Required software

1. JFlex
2. Java/JDK
3. CUP/JCUP

Limitations
Supported files

1. .cm cminus files

Improvements

1. Further cleanup of semantic analyzer code
2. More detailed error reporting
3. more documentation

Contributions

Ben

1. Created Symbol package
2. Created Symboltable.java
3. Wrote semanticanalyzer code
4. Debugging
5. created documentation

Conor

1. Worked on semantic analyzer code
2. Linked semantic analyzer to main
3. Linked semantic analyzer to cup
4. documentation

Acknowledgements
This project was heavily based on/inspired by resources provided by Dr. Fei Song's compilers content. We were provided a rough idea of the implementation process as well as a portion of starter code. The grammars created in CUP are heavily based off the the C- specs. The absyn class structure is heavily based off of lecture content.