Checkpoint 2 Report

Introduction

The major focus of this course is the Implementation Project, wherein we are tasked with designing and developing a fully functioning compiler for a simple procedural language. Checkpoint 1 of this project had us design and implement a Scanner to generate a sequence of relevant tokens, a Parser to generate and displays an abstract syntax tree, and an Error Recovery System to report and recover from errors during the parsing process. Checkpoint 2 involved designing our program to be able to detect and report semantic errors such as mismatched types in expressions, undeclared/redefined identifiers, and so on.

Accomplishments

We were able to fully implement both required modules of checkpoint 2. The Symbol Table has implemented the following data types: Hash tables, Simple variables (int & void), array variables, Functions/Blocks (-entry and exit), and Errors (undefined/redefined). Our Type Checker is able to recognize the following: array range/index are of type int, both sides of an assignment, both sides of an operation, function calls and return expressions, test conditions must be of type int.

Related Techniques

The development techniques utilized by our team for this checkpoint compare similarly to our previous checkpoint. Such techniques include time management, division of workload, breaking down sections into smaller pieces, and rigorous testing of each section (especially with type checking). We made great use of the techniques offered in the lectures, which includes using Nested Scopes, Name Bindings, and Type Conversions/Coercion. Our program also makes use of post-order traversals in order to move through the abstract syntax trees. As we did in Checkpoint 1, our group took advantage of Pair-Programming in order to quickly develop and bug-fix our program.

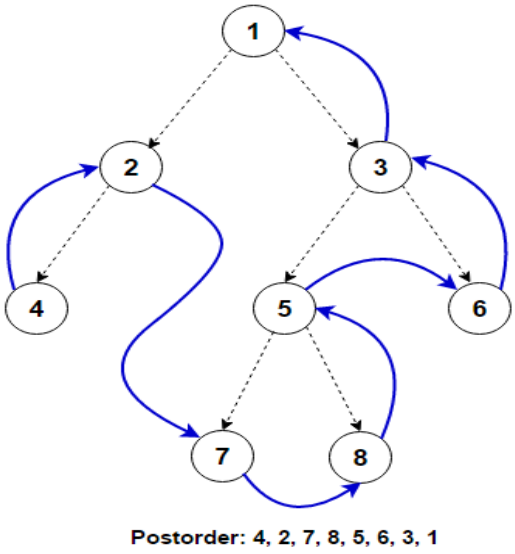

Figure 1 Post-order Traversal

Design Process

Our design process for Checkpoint 2 strictly adhered to the lectures and textbook, with some clarification from internet based resources. We first began by analyzing the assignment and broke it down into smaller sequential problems. From there we began to develop each module in order.

The first module was the Symbol Table. Breaking it down into its main components we have: a hash table with an arraylist of a type preserving class added for this checkpoint. It is able to keep track of scopes, as well as manage itself on scope changes. The second module was the Type Checker. Its main components are: checking array range/index is int, checking both sides of an assignment, checking both sides of an operation, checking function calls and return expressions, and checking that test conditions are ints, as well as various other semantic considerations to ensure sane program output in the future. Finally after everything was implemented we began a testing routine to insure everything functioned as it should, and that errors reported/recovered correctly.

Lessons Gained

As we developed Checkpoint 2 of this project we were able to gain some further insight into how our Checkpoint 1 interacts with Checkpoint 2, and what that means in the broader scope of a compiler, wherein the Scanner/Parser feed into the Semantic Analyzer. Touching on a lesson we learned from Checkpoint 1, we decided early on that it would be best not to deviate from the program structure given to us in the lectures, as this had caused us many issues previously. This ended up being a very good idea as things went more smoothly than before, and will be our go-to approach for Checkpoint 3 as well.

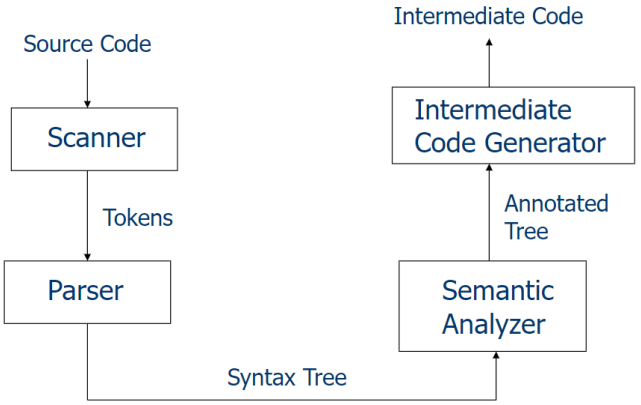

Figure 2 Front-end Compiler Design

Assumptions & Limitations

There are some assumptions we made during this checkpoint, as well as a few limitations we encountered. An assumption we made during this Checkpoint was that a Hash-Map of ArrayLists (HashMap<ArrayList<Type>>) was the best route to take because it was talked about by the Professor. There may of have been easier routes to take but we made the assumption that the Professor intended for Checkpoint 2 to be designed this way, however scoping proved to be a challenge and more consideration should have been given to that aspect given its importance and how troublesome it was to debug. In regards to limitations, the given class structure was difficult to work with but not impossible, but it did require some trial and error in order to have everything work as intended. A final limitation is that down-casting in Java is especially painful, taking up a lot of space and generating peculiar errors during our development of the Checkpoint. Much time was lost to such errors but in general it was not too harsh.

```
public SemanticAnalyzer(StringBuilder lol) {
        this.table = new HashMap<String, ArrayList<NodeType>>();
```

Figure 3 HashMap<ArrayList> implementation