CIS*4650 ~ Project Report

Isabella McIvor (1101334) , Huda Nadeem (11439431), Zuya Abro (1109843)

What has been done for this checkpoint:

This checkpoint we completed making a correct cm.flex file that reads in the test file (.cm) and prints out accurately to the language. When writing this in the terminal we were able to confirm it was being scanned correctly [java -cp /usr/share/java/cup_runtime.jar:. Scanner < ./tests/fac.cm > output.txt]. Moving on to the cm.cup file, we included every c language specification, and compiled [Java -cp /usr/share/java/cup.jar:. CM tests/fac.cm] with 2 warnings, no error. One of the warnings was for error handling which we hadn't completed at this checkpoint. We created .java files according to the terminals in the absyn folder one by one when updating the cm.cup file. When it came to testing against the cm.cup file we had a common issue with each .cm test file where the iter_stmt (non terminal) wasn't implemented correctly in the cm.cup file. We did extensive error checking and debugging but encountered a challenge causing a halt in our progress, leading to little error recovery time. We were able to accomplish a generating AST using CUP with limitations we will mention later.

Design process:

1. **Preparation**

The first step of this checkpoint was to study the SampleParser by recognizing its techniques and behaviors, as well as identifying every separate piece that needed to be created. The pieces that we listed to be created were the following: our JFlex specification, cup specification, Scanner class, ShowTreeVisitor class, absyn classes and CM class.

2. **JFlex Specification**

We started the coding process with the JFlex specification that is needed to make the scanner. The first step when creating the JFlex specification was to identify the keywords and literals that were needed to define all symbols. The process of this was very simple as it did not require much change from the example parser. After defining all regular expressions, keywords and literals that would be used to represent symbols.

3. **Cup Specification**

When creating the cup specification, we first defined all of our terminals, non terminals and the set of simple grammar rules from the C- specification, not including any results. Next, we defined all the classes in the absyn file that represented the terminals using the Class-Based AST for C-minus from the lecture slides. We gave each class a constructor that set its attributes and a visit method to be used when creating the abstract syntax tree. Choosing this as the next step was beneficial as it made it easier to develop the simple production rules from the grammar by choosing which objects to create when providing each rule with a result based on the matching absyn classes. After giving the results to each non terminals production, we made a list planning which type each non terminal would be (i.e. which absyn class would be assigned to each terminal). An example of this is changing a declaration “non terminal type_spec” to “non terminal NameTy type_spec”, as a typespec non terminal is defined as a NameTy object. It was helpful to create this as a written list outside of the program to plan and organize which non terminals connected to each other based on their types. The goal when assigning the variables to absyn classes was to make sure that the object type of the production rule’s result was either the same type or inherited from the object type of the non terminal.

Summarizing any lessons gained in the implementation process

- ➢ When developing we wanted to translate and create a proper cup file so that the file was accounting for every specification given for C. A challenge we came across was the structure not checking for every case scenario, especially those leading to shift/reduce conflicts in parser generation. We learned to be very clear about grammar rules so that there isn't any ambiguity.
- ➢ Iterative testing and refining code were a huge part in uncovering issues and ensuring the reliability of the parser. We learned when updating the code to debug, it’s hard to keep track of changes.
- ➢ This leads to version control, we all worked on the project alone and added parts to each. This was extremely disorganized, and a git repository should have been made to collaborate better.

Discussing any assumptions:

- ➢ We assumed a recursive parsing strategy, when going through the cm.cup file you can see each non terminal is connected with the previous one, we created a structure in the file from top-down.
- ➢ We assumed the listed syntax rules given in the C-specification files were all the one we needed to create rules for C-language in our cm.cup file; we went according to those specific rules.

Limitations:

With our test files, we found a common issue with parsing the while loops. After running and testing with certain files that had a while loop, the program would output a syntax error which consequently could not “repair and continue to parse”. This should be addressed before moving forward with checkpoint 2. Additionally, we have limited error handling. We have

created test files that introduced errors; however, we have not fully implemented the error handling for these issues (error recovery).

Possible improvements:

- ➢ Parsing: There were many cases we didn't account for, checking how to handle operations within if statements and while loops, common cases in c language specifications.
- ➢ Error Handling: There could be more tests and recovery checks we could do in order to guide the user for a better experience.
- ➢ Address warnings: 1. Terminal "Error" declared but never used 2. Shift/Reduce conflict found in state #83… resolved in favor of shifting

Contribution/Assessment:

Ella:

- ➢ Collaborated on creating the cm.flex and cm.cup file
- ➢ Collaborated on the documentation
- ➢ Created test cases for the parser

Huda:

- ➢ Collaborated on creating the cm.flex and cm.cup file
- ➢ Collaborated on the documentation
- ➢ Error checking to solve why the AST was not printing

Zuya

- ➢ Collaborated on the documentation
- ➢ Collaborated on finding errors within the cm.cup file
- ➢ Collaborated on testing