Implementation Project – Checkpoint One

March 5, 2018

Jeremie Fraeys - 0892019 and Joel Klemens – 0895223

Review of implementation process:

When we first started working on this project we needed to decide if we were going to use C or if we were going to use java. Due to the fact that we had both completed the introduction assignment in C it was decided that this would be the best language to follow as we both felt more comfortable in Lex / Yacc oppose to flex / CUP. We then decided that we would use the 'Tiny' example that was provided by the instructor. This was very helpful, we were able to learn about the scanning and parsing system by reading through that code. Using the file structure that was outlined in the tiny scanner/parser we were able to start to adjust the variable types, as well as the Lex file and the Yacc grammar to fit the C minus language.

As the first part of the semester and the first assignment were dealing with regular expressions and Lex we started our assignment with changing the file "tiny.l" to "cminus.l". To make the lex file fit the C minus language we had to add all of the special symbols that were outlined in Appendix A, "Lexical Conventions of C-".

Next, we had to change a couple of other files simultaneously to be able to match the new grammar that would be outlined in "Cminus.y", updated Yacc file. This entailed adding and removing a variety of different variables and structure types in the files "global.h", "util.c" and also the "makefile". The names of these files have also been altered to more appropriately fit our implementation.

In "global.h", we had to add an enum type "DecKind" which we learned we would need from following the outline of the grammar from the specification file as well as from the lecture notes (6-SyntaxTrees, page 10). This is because in C minus we would need to account for declarations of variables, functions and also parameters. To finish off this section several enum types were exchanged to fit the specification and everything pertaining to tiny was removed.

The file "util.c" also needed to big adjustments in order for it to fit the C minus specification, however, most of these changes were along the lines of adding, removing and renaming different variables to account for the special symbols in C minus. The big changes that needed to be made were editing the function printTree(); again to fit the C minus specification and as we had added the enum DecKind in "globals.h" we also needed to add a new TreeNode type that would save the declaration nodes for the syntax tree.

Very little had to be done in the makefile other than changing the updated names of the files and adding some special cases for our own testing purposes. This is because from the tiny specification all of the files were already covered with individual compilation cases.

Finally, the most complicated part of the assignment was the Yacc file for the grammar specifications. We had a lot of troubles with initial implementation of the grammar. To make things easier we started with writing out all of the semantics by hand and trying to apply a list of pseudocode based off the way the grammar was coded in the original grammar for the tiny language. Again, we had to swap out the special characters and add some static variables necessary for our new data types. Using the lecture slides along with the reference manual for Yacc we were able to outline all of the grammar rules. To fit the outline of how to build syntax trees from the lectures we simplified all of the expression rules into one called "exp" instead of using *expression, simple-expression and additive-expression.* We also had problems with getting our syntax tree to handle the different variable declarations with the type specification as there is no type Kind. To deal with this problem we met with the professor who suggested that for a declaration we should create a new DecKind node in order to store the types “int” and “void” in the attribute name field. Then in the case that we would have to also store variable names or array ranges we would then attach “IdK” for the name or “ConstK” for an array range as a child of the declaration node. After doing so our syntax tree was then able to handle all types of declarations defined in the grammar semantics.

What has been completed this checkpoint

After checkpoint number one our C minus compiler is able to use the grammar semantics and regular expressions to scan through C minus files. Output is created from the individual tokens along with their location within the file, syntax errors can be caught and defined in this process. Then the grammar of the C minus file will be outlined by different statement, expression and declaration nodes containing names and values.

Possible Improvements:

In any program there is always room for improvement. These improvements for our could possibly be made in the Yacc file as we become more comfortable with this parser. We were able to make improvements by simplifying many of the rules, but we feel this could possibly be improved in the future.

Even if though the compiler may work with a simple and correctly written C minus file the bigger challenge is to have the parser catch all of the possible syntax errors while also being able to recover properly.

Individual Contributions:

For the majority of this project we worked together using pair programming so that we were able to discuss options and implementation techniques in person and apply these to our project together making sure both of us would be equally familiar with how the scanner / parser would work. Even though we worked together on most of the more difficult parts of the assignment, individual tasks were assigned. Jeremie completed most of the work on updating the files “util.c”, “global.h” and “cminus.l”. Joel completed most of the work on the initial set up of “cminus.y” and the documentation. However, most of the difficult implementation problems were handled as a team using pair programming. Throughout this checkpoint each of our work was backed up to personal branches of a project github repository and communication was accomplished using a private Discord channel.