Project Report

Checkpoint One

# Kyle Monteiro - [kmonte04@uoguelph.ca](mailto:kmonte04@uoguelph.ca) -

# Frank Maciel - [fmaciel@uoguelph.ca](mailto:fmaciel@uoguelph.ca) - 0999850

## What has been done for this checkpoint?

A grammar was written out meant to map the C- language according to specification. The scanner for the C- language was then created using JFLEX.

Wrote out the set of productions for clarity and better understanding. Created the parser for the C- language.

After creating the C- parser we Implemented error handling for semantic errors.

At this point we decided to create the test files showcasing our error handling. Here we also performed further testing into our error handling for certain cases.

We realised that we would be building on current code, and so in planning for later features we decided to implement flexible abstract classes.

## Related techniques and the design process

The idea was to first have our scanner and grammar working before thinking about error handling as well as compilation errors and shift reduce warnings (if possible).

We started by working on the scanner. The grammar was created, and the scanner was worked on with respect to the grammar (the JFLEX file included in the C1 package was used as a basis).

Afterwards we wrote out the productions provided to get a good gist of how the tree is parsed; then used this to create our parser (the CUP file included in the C1 package was used as a basis). We created the parser from top to bottom from the C-Minus spec; as new objects were needed, the class would be created and productions would continue being created. After the productions were completed we decided to look at the compilation errors and resolve any outstanding errors.

After compilation errors came the shift reduced warnings that we had to resolve. Finally after resolving the issues, we decided it was time to implement error handling. We implemented error handling by introducing one error at a time and then testing it. Once everything seemed according to specification we started to look at the organisation of our code as well as cleanup that was necessary. We cleaned up dead code, old comments, and efficiently organised code for later modification.

Finally, testing. Testing occurred throughout the entire checkpoint completion. At every stage we would consistently test our scanner with the provided code, and the parser with the same code as well as later on test error handling while working on test files.

## Lessons gained in the implementation process

The implementation process gave us a lot to learn from. One of the biggest lessons is most certainly the one of incrementally building a piece of software. There was no way to complete this checkpoint by building various things at a time; you would get lost! By building this project a piece at a time, and learning the components at each step, we efficiently completed this project.

Implementation of interfaces and abstract classes can add further value by catching errors before compilation. This helped sort out our errors prior to compiling the code and catching it later on.

Clean and organised code will allow for an easier time building in the long run. It was tempting to use the quickest names for productions as well as terminals, but by giving ours clear names, we quickly realised the benefit going forward.

Finally, by taking the time to go through the grammar rules and productions on paper, we allowed ourselves to gain further insight into the language we would be working with, which made our coding much more efficient.

## Assumptions, limitations, and improvements

The c-language assumptions come from the specification file. There weren’t necessarily other assumptions made for the assignment; aimed to follow specifications provided.

We did find it difficult to implement error handling for recursive functions. The function declaration parses after the function call which throws “function not defined” errors. Some improvements we would like to make going forwards is the handling of more complex errors in the next checkpoint, as well as making changes to array lists which hold functions and variable declarations to not be of type String in the future. We would also like to include some more helper functions with the purpose of reducing reused code in the cup file.

## Team member contributions

Both team members essentially collaborated on the entire project, working together for each aspect of the project. The team consistently met up and worked on the project; swapping roles while peer programming (programming vs analysing). We completed the scanner, parser, error handling, as well as creating test files in this manner and enhanced our communication and team working skills here. We also organised discord meetings for when it came time to work on the document, using these meetings to reflect and bounce thoughts off of one another.