**Testing**

Testing is incredibly important in a software development project of this size. Test-driven development would have been perfect to accomplish the creation of this compiler – if the students had all perfectly understood the process of its creation from the very beginning. However, this obviously wasn’t the case, so we can immediately rule out test-driven development from the beginning. However, careful and accurate testing, of both negative and positive types, is important. Negative and positive testing means that testing took place that expected and did not expect failure on compilation.

Testing of the various parts of the compiler occurred in different ways. The testing of the scanner took place through unit testing – it basically tests that the tokenized output created by the scanner is as expected, and the testing is specific to a specific mini-Pascal file.

The testing of the parser was done through creating several different mini-Pascal files – about six happy path files and six sad path files. The happy path files were expected to run correctly and became more and more complex with each file iteration. The sad path files did not necessarily vary in complexity, but instead in terms of the type of error they caused.

In the case of the syntax tree, manual checking of the correctness of the tree was used until I was satisfied that it was correctly generating. With the more basic mini-Pascal files, it was much easier to determine what the tree should look like, so in the case of the more complicated pascal files this method may fall short.

In the case of the code generator, outputted assembly code was run through QtSpim, one of the most widely available and used MIPS assembly emulator. QtSpim would detect any errors, and their genesis could be traced back from there.