CS325 CW1 Report

# Design:

The parser is using the recursive descent algorithm and the general steps it takes to parse a file are: use shlex to tokenize the input file, keep trying to find blocks of code while the current token is not the end of the file finally try to parse these blocks by implementing the syntax outlined in the lua manual. It replicates this grammar by first looking for any keywords e.g. ‘do’, ‘while’, ‘local’ and then it will call the appropriate sub parsers for these keywords. There is a sub parser for every bit of grammar as outlined in the lua manual. These sub parsers check that the next tokens match up with the lua grammar the parsers can also call each other when appropriate e.g. a var parser can return a: prefixexp, ‘ [‘, exp and a ‘]’ parser. For conditions where the parser’s next token is not a keyword e.g. the line ‘callingafunction(x)’ the parser will try and find different possible options that would be valid code and then commit them and return them if they don’t return any errors.

Should a parse function fail the parser needs to be able to continue and continue parsing the rest of the script. To accomplish this the parser will print out the error it’s encountered and then simple ignore that token exit out the current parse function and then let the program carry on trying alternative parse methods.

# Difficulties:

Bug testing was the largest difficulty faced as due to the program using recursion. I found the program would often get stuck in loops where function a would call function b and then function b would call function a etc. Print statements also gave limited information as it was often hard to know whether the looping functions were the issue or potentially a parent function that called them.

A function named copylexer was also created when passing variables to python functions the parent function’s variable won’t be updated when assigning value ie function a calls function b with variable lexer if function b performs ‘lexer = some value’ the variable lexer in function a won’t be updated. Copylexer allows function b to assign a lexer with a value and update the parent’s variable. It accomplishes this by using the using the shlex instance’s methods get\_token and push\_token by clearing the target lexer and pushing the src lexer into the target.

# Testing:

To try and keep a constant eye on all parts of the code I wrote some unit tests that would test each of the parse functions explicitly. Due to the nature of the parser returning a string of the whole input code on a successful parse I could feed in valid lua code as an input and confirm the parser returns the valid code. For example, if I wanted to test that lua parameters where being parsed correctly I would specify some valid parameter inputs call the parseparlist function store the result and then confirm the result matched the expected result.

Several tests were created to test all the parse methods in the project and they could then be ran whenever I made modifications to the code so as any parts that were broken would be easily identifiable.