Lynne Coblammers
EECS 665
Lab 4 Report
2015.09.16

Speaking generally, C-like languages treat whitespace identically, regardless of how it's created. One tab or four spaces, it matters not. This is not the case for many other languages (Python, Haskell, YAML, etc.), where the construction, and placement, of whitespace is important. What complications does this introduce to our lexer if we need to account for such concerns? As a hint: consider writing a lexer for a Makefile, where only tabs and not spaces are allowed. How do the rules you have written change?

Accounting for placement of whitespace would make writing the lexer rules more nuanced. For example, if a lexer were reading a Makefile, it would need to look for a newline followed by a tab, which would denote the end of a dependency list and the beginning of a command to be executed. If there are instead spaces or if there is no newline, tab, command pattern seen, an error would need to be reported. In C and other similar languages, program structure is primarily determined by things such as parentheses, brackets, and braces. White spaces can be treated equivalently, just marking the boundary between two tokens. In the simple lexer created for this lab, this means all types of white space could be read in, treated equivalently, and essentially be ignored. However, in Makefiles or in languages where white space has meaning, it has to be read and interpreted by the rules and can't be ignored, glossed over, or treated equivalently. This might mean writing separate rules for newlines, tabs, and spaces. In reading a Makefile, one possible rule could be to enter a new state when a newline followed by a tab is seen. In a translation, specific types of white space might need to be translated into other symbols such as braces or parentheses.