Andrew Johnson joh11879
Wyatt Kormick kormi001
CSCI 3081W
Group Archaeopteryx
Writing Assignment 1
3/24/17

Our iteration 1 code had a few inefficiencies present within it. While not exactly evident on such a small scale, these inefficiencies would be disastrous on a large scale. These problems in our code would probably cause memory issues and make the program take a long time to run. Our code only contained three of the common errors present in many iteration 1 projects. Our project called make_regex many times, every time the scan function was called. It had a dependency on the ordering of the token types, which are prone to change, reducing the scalability of the project. It also redundantly defined a large amount of regex_t pointers before putting them all inside of a regex_t pointer array.

A problem that appears frequently among iteration 1 projects relates to inefficiencies in the scanner_tests.h file. Most commonly are tests calling the make_regex function. This is inefficient because the regular expressions have already been made when the scanner object get instantiated at the beginning of the tests, and doing so again is redundant. Another potential problem could be if a new scanner object was instantiated in every test. Only one scanner object needs to be instantiated, and then each test can call upon that scanner's scan function. Our code did not have any of these problems.

Another common inefficiency was having the regular expression definitions inside the scan function. In this way, forty or so calls to the make_regex function would be made every time the scan function was called. This is very inefficient, since regular expressions will not change between calls to scan. Each regular expression only needs to be made once for each instantiation of the scanner object. This problem occurred in our code in src/scanner.cc on lines 40-93. We had all of our regular expression definitions inside the scanner function. We fixed this problem by moving the all of the definitions out of the scan function and into the scanner class itself, so that they are all made only when a scanner object is instantiated.

Having an array of TokenType values is useless. The point of making such an array would be to call the TokenTypes by an index value, but the TokenTypes already exist inside an enumeration data structure. Enums are capable of being indexed just as easily as arrays. An array for these values would just be redundant. Our code did not have this problem. We indexed our token types using the enum.

If the order of kVariableKwd and kEndKwd were switched around our code would encounter problems because we have a separate array containing all of the regular expression constants and it is crucial that the order of the regular expressions matches the order of our enumerated token types. Prior to fixing our code this error occurs in lines 187-230 in scanner.h. This is an issue because if the order of our token types change, the scan function will no longer work unless we change the order that is defined

in our code. It would add a level of difficulty with manipulating our code as it would require more work to fix. To fix this issue, we now index the array of regular expressions by referring to the matching index of the token type.

Our code creates a new regex_t pointer for each regular expression before putting it in the regex_t* array. This occurred in scanner.h lines 135-185 before the fixes. This is a problem because it is unnecessary to define all the regex_t pointers before putting them in the the array of regex_t pointer type. To fix this issue, we changed our code to have the regular expressions defined inside the array in the scanner constructor in scanner.cc. By doing this, we can skip the process of defining each regex_t pointer and do half as much work as it would have been to define and add to the array.

Our code did not contain any places where integer literals were used rather than enumerated TokenTypes. However, this would have been a problem because inclusion of numbers such as integers lead to less malleable code. Every time you change the index of the enumerated value you would have had to change the value of the integer used. Using the integer token type instead would save you from unnecessary work down the road.

The changes to our program made our code easier to work with and more flexible and slightly more efficient on a large scale. Problems like having all the regular expression definitions outside of the scanner class and instantiating all of our regular expressions outside of the array would cause unnecessary operations to occur. By fixing these problems, we save runtime. Fixing issues related to the order of token types gives us greater freedom to manipulate the code without having to backtrace and rewrite certain portions of it. The improved efficiency may not be evident on our exact code, however, on a greater scale improving efficiency would be very beneficial.