Writing Assignment #1

By Frank Bi (#3804098) and Michael Cistera (#4352091)

Introduction:

Of all six possible problems that are required to be fixed, our scanner implementation only had problem number four. The five other suggested fixes were around reducing redundant code and improving efficiency if we ever needed to change the regular expressions or the token types.

Problem four brought up the possibility of code breakage when the order of the enumerated types is changed. Before improvement our code would not function properly in this case.

Q1: Is there any duplication in the testing code you've written to test individual token regular expressions and the code used by scan?

There is no duplication. The test code does not make the regexes itself, instead it makes one scanner and uses the scanner's regexes repeatedly.

If we were to make a regex in more than one place, if we update it, we would need to make the change in more than one place.

Q2: Does your scan function make calls to makeRegex every time it is called?

Our scan function does not make a call to makeRegex every time it is called (scanner.cpp line 31). We avoided this inefficiency by creating an initialize_regex function that calls our add_regex function instead. This function is called only on construction of the scanner. After the initialization, each regex is stored in a vector.

If we were to make a call to makeRegex each time scan is called, besides being unnecessary, the only real problem it would present simply be wasting resources despite everything running correctly.

Q3: Do you have a redundant array of tokenType values?

We do not have a redundant array of tokenType values because we declared tokenType with typedef (scanner.h line 49).

An array for tokeType would be useless to begin with because if we were to reference each type in tokenEnumType in an array, we would have to have an index into the array to access the value, and the index is the same as the value. Using an enum instead, we wouldn't need an array index to access the value, we would simply be able to reference it by its name. This does not limit us because we can still move sequentially through each regex in the vector. (scanner.h lines 74-75)

Q4: Would changing the order of nameKwd and exitKwd in the definition of enum tokenEnumType in scanner.h have any adverse effect on the rest of the code?

Yes, changing the order in the enum tokenEnumType would break our code on lines 148 and 186 in scanner.cpp. First, on the loop on line 148, the for loop was originally looping from nameKwd to notEquals. On line 186, there was originally a loop from whiteSpace to lineComment. If we switched nameKwd and something else, for example, we would start from a different position, therefore skipping over some of the previous parts. If notEquals ended up before nameKwd in the enum it would end up in an infinite loop.

What we did to fix this problem was first to split up the regexes into two vectors – regular regexes and the white space regexes. The new loops loop through the vector based on the vector's size, which solves our problem from above.

However the test code expects the regexes in a certain order – the same order the regexes are listed in the enumerated tokenEnumType. To resolve this, we created the add_regex function (scanner.cpp line 79) that takes a regex string and places the regex in the slot of the vector corresponding to the provided tokenType value. In this way, the order in which we add the regexes does not matter. The test code now works regardless of the order we put them in.

Q5: Do you create a named regex_t pointer for each regex instead of only putting them in an array?

We used a vector instead of an array so that the size could be dynamically allocated.

If we were to create a named regex_t pointer for each regex it would be incredibly tedious and inefficient.

Q6: Are there any places in which enumerated tokenType values should be used instead of integer literals?

We use enumerated values and not integer literals. If we did use integer literals, if we were to change the order of the enums, we would have to manually change the integer literals by hand.

Conclusion:

The changes we've implemented has made our code more flexible. Regarding problem four specifically, we no longer depend on the order of the enumerations which makes expansions easier.

The single change to our code does make an improvement on our scanner, but as a trade-off the extra function add_regex does bulk up the code and increase the complexity slightly .