Writing Assignment #1

By Frank Bi and Michael Cistera

Introduction:

Of all six problems listed that are required to be fixed we were efficient in that only problem number four needed to be fixed to improve our scanner implementation. 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 instance of when the enumerated types are listed out of order, which before changes, would have broken our code.

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. It does not make the regexes itself, instead it makes one scanner and uses it repeatedly throughout the testing code.

If we were to make a regex in more than once place, if we update it, we would need to make the change in more than once 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 scan is called (scanner.cpp line 31). We avoided the inefficiency by creating an add\_regex function that calls makeRegex instead. The regex is initialized when the scanner is initialized. 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 compiling 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 while 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.

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, there were originally a loop 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 which could potentially create an infinite loop.

What we do now is splitting up the whole regex as two vectors – regular regexes and the white space regexes. The new loop loops 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 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 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 if the enumerations which makes expansions easier.

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