Group: Turtwig 3/24/2017

Improvements of Scanner Code

Introduction:

This paper details the process of improving scanner code efficiency by locating and removing redundant code segments and through restructuring existing processes. The original version of our scanner code had several important inefficiencies which have been removed. These inefficiencies corresponded with lab 8 problems 2,4,5 and were calling the makeRegex function every time a scan was conducted, incorrect structuring of token types due to indexing, and creating redundant pointers for each regex.

Explanation and solution:

1. The original version of our iteration 2 scanner had no duplication in testing code for the token regular expressions as we only create the regular expressions once upon initialization of a scanner and do not repeat this process our tests. Having calls to make_regex within the tests written in scanner_tests.h would be inefficient and redundant as the regex already exists within the scanner itself and can be tested through comparisons to appropriate lexemes. In our code these comparisons are checked by being passed into a helper function named tokenMaker_tester.

The function scan_token, which is renamed to decide_terminal in our code, should not instantiate a scanner each time it is called as part of scanner instantiation is creating all the necessary token regular expressions. By creating a new scanner every time scan_token is called the function takes more time and occupies memory space. The current design can be improved by passing an array of the regular expressions and their corresponding tokens to the scan function which would instead be created upon instantiation of the scanner.

2. Initially the scan function would call make_regex every time a call to scan was made as seen in Scanner.cc line 216. This was inefficient as the regular expressions needed for every scan are the same and recreating them adds unnecessary overhead to the scan process. To prevent inefficiencies resulting from having to repeatedly create all of the regular expressions we simply call the make_regex function once for each necessary token case within a helper function called MakeTokenRegex. The larger redundancy issue was fixed by putting MakeTokenRegex into the scanner's initialization rather than within our scan function. The array of regexes are created by MakeTokenRegex is then passed to the scan function as a parameter, allowing the overall design of the scan function to remain unchanged from the original.

3. We never constructed a redundant array in our iterations as all of the TokenType values are already in an array along with the corresponding regular expressions. Since we must keep track of both the type of token and its regular expression, having a separate list of just TokenType is simply inefficient.

4. Changing the order of kVariableKwd and kEndKwd in the definition of enum kTokenEnumType in scanner.h would cause an error in the original version of our scanner. In the old version we made an array of regex_t*, in which every regex_t pointer corresponded to the index of every definition of enum kTokenEnumType. The original code can be found in Scanner.cc line 270-273. For example, in the definition of enum kTokenEnumType, kIntKwd is first, so the regex of kIntKwd in array of regex_t pointer will also be the first. Therefore, once we know the index of regex_t pointer in the array of regex_t

pointers, we can also know the position of its corresponding kTokenEnumType in the definition of enum kTokenEnumType. Thus, if we change the order of definition of enum, the order of regex array will be incorrect and the result will also be wrong. This problem should be corrected so that we can access the appropriate token, regex matchings without needing to know their arbitrary index of the original listing.

To fix this problem, we created a struct regex_helper which contains regex_t* regex_ and TokenType terminal_type. Every regex_helper variable has a fixed regex_t pointer and a matched kTokenEnumType to its regex_t pointer. For example with kIntKwd, the scanner would begin by first creating a regex_helper of kIntKwd and setting its name it as IntKwd. Its regex_t pointer will be the regex of kIntKwd and its TokenType also will be set as kIntKwd. If the text is matched with the regex in regex_helper of kIntKwd, we can easily know the kTokenEnumType of kIntKwd is kIntKwd through calling IntKwd.terminal_type. Therefore we can make sure that once we find the corresponding regex_helper to the text, we also can get the regex_t pointer and TokenType of the text. Ultimately the regex_helper structure enabled us to fix the 4th problem.

5. In our original scanner we created named regex_t pointers for each regex and then put them in array of regex_t pointers as found in Scanner.cc line 67-171. This was unnecessary and redundant because we could simply just set each element in the regex_t pointers array directly rather than separately. Thus we delete all of the named regex_t pointers and set each element in the regex_t pointers array directly. This fix improves code readability and simplifies the array construction process and reduces memory overhead.

6. We do not have places where enumerated TokenType values should be used instead of integer literals. We identify any keywords in enumerated TokenType by finding their matched regex_helper. Once we find the corresponding regex_helper we then can use our regex_helper struct to get the TokenType of the regex.

Conclusion:

In the new version of our scanner we improve the efficiency of our program by solving problems 2,4,5 as described in lab 8. Corrections to the scan function fixing the issue of repeatedly calling MakeTokenRegex saved valuable memory space while restructuring of the token and regex data structure dramatically simplified the storing and accessing of this important data. The most valuable fix from this process was creating a two dimensional array in which to store our regex and tokens, enabling dynamic accessing independent of original indexing. The overall fixes to the code had no major implications on the functionality of our scanner but they did relatively quicken the runtime of the scanner and reduced memory usage of the scanner.