Name: Haris Sohail
Date: 05/02/2018
CS362 - Random Testing Quiz

In a truly random situation, this test would require a lengthy amount of time to run. For example if inputString() generated based off all 128 characters, making the word reset perfectly would be quite difficult. This does not even include and unfixed string size. To ensure this random test was possible I had to put in a few constraints, though some functions truly worked as random fine due to a wider acceptance criteria.

Starting with inputChar(), which I programmed as fully random. It can return any ASCII character when called. It picks a random int between 0 and 127 and assigns the char to that value, which represents the ASCII character. That char is then returned. The reason why I left this function full random which constraining the other is because there are only 128 possible returns, when called 1000+ times this results in very likely getting all of those values returned eventually.

Now inputString() required some fixation in order to ensure a speedy end to the tests. First I set the return string to a fixed size of 6. This is so when it is time, the word reset has a higher chance of being produced. Next I placed each character in “reset” into a char array. Characters from this array are randomly assigned to the result string. This means repeats can occur, such as having a string of “eeeeee.” This results in truly random returns all the while guiding the program to eventually read reset.

The results indicate 100% line coverage for testme(). This means every line, statement and branch are being executed as per the gcov file. The only line not executed in the entire program is the return 0 in main. As the only way to break the loop in testme() is to hit the error which exits the program. The only issue with this program being, is the if statements can only be tested once per run. In terms of it evaluating to true. Ideally I would alter the testme() function to return to main once reset is called, at which point testme() could be called repeatedly to ensure multiple tests per statement. Overall this random test generator required a small level of guidance to ensure a proper test procedure.