Methodology

When approaching this problem, I decided that the easiest solution would be to deal with ASCII characters for my input pool. I figured that this would provide a streamlined way to get a char from a randomly generated integer and thus also allow me to build random strings quickly. At the end of testing, I believe that this was a good choice for the project.

As for the actual methods of inputChar and inputString, I feel that they were overall quite simple but nonetheless effective. In terms of inputChar, initially, I simply wrote a line to return a random integer between the bounds of Space and the closing bracket and then cast the returned integer to a char. This worked great on its own but got a little refinement down the line when I realized that I wanted to dynamically impose the limits so that it could be more useful. Thus, I added a number of input parameters to represent the upper and lower bounds of the desired set and passed those to the random number generator. This change came into play because when working on inputString, I chose to use inputChar to return chars with which to build my string.

The only real issue that I had with testing this was that I had to work a bit to fine tune input pool so that the probabilities of turning up a string that equaled ‘reset’ would be manageable. When I first tested the program, I had not yet changed inputChar to take limiting parameters, and it took far too long to actually find the “bug”. In fact, when I ran this first test, I let it go for over 47 million iterations (about 20 minutes) and I decided to terminate it instead of letting in run to eventually find it. Then, once I had changed inputChar, I tried limiting inputString to only capital and lowercase alpha characters and it still ran far too long. Next, I took the step that I thought would be enough to corner the bug by limiting inputString to only lowercase alpha characters, but I was still unable to reliably turn up the bug in under 5 minutes, although some tests did. Finally, to ensure that the program did not run too long, I limited the string generator to only the characters that are encompassed by ‘e’ and ‘t’ (the lowest and highest ASCII characters in “reset”) and was able to average a runtime around a minute and a half.

Finally, I also added the ability to time-keep into the testme method so that I did not have to sit with a stopwatch to ensure that the program was finishing within the allotted time.