Dennis Trinh

CS 362

Professor Aburas

July 22, 2018

Random Testing Quiz

**Code Analysis:**

The goal of this quiz was to successfully get to the error message utilizing random testing in a reasonable amount of time (~ 5 mins). The first thing I had to figure out was what exactly was being tested. The main part of the testme function is a while loop with a bunch of conditional statements. At the end of the set of statements, there was the error message. In the set of conditional statements, there seemed to be two separate sets of conditional statements: one set included nine conditional statements that checked for the current character and state of the iteration and one that checked the contents of the string to see if it matched "reset". However, to get to the second set, I would first need to pass the first set of conditionals. This is my starting point.

**Implementation:**

To first get through the first set of conditionals (getting to state == 9), I had to first design the inputChar function. At first, I had a fixed string containing all the necessary characters to satisfy the first set of conditionals.

The array looked something like this:

const char symbols[] = "({[ ax]})";

I picked randomly from the string and returned that value as the character picked. I found I would get to state 9 in a fraction of a second. I decided to try the entire set of ASCII characters from space to tilde and still discovered that state number 9 would be reached very quickly.

The next part was dealing with the string. I had to find a way to effectively get the string to get the word "reset" while still keeping the random testing aspect of the quiz. To save time, I set the string to a fixed 6 characters (5 characters + a null terminator). I also tweaked the function to accept and modify the string and return nothing. When I tried to generate a random string using the original char * function header, I ran into some segmentation faults.

In my first attempt of implementing the inputString(), I tried all the ASCII characters. The loop went for more than five minutes without ever generating the appropriate string. The probability that those five characters would line up is way too large. I cut it down to all lowercase and uppercase letters and still found that the error message would never be found in time. I then cut it down to just lowercase letters and found that a few runs would be within 3-6 minutes, but there were certain runs that went on for much longer. In my final iteration, I cut it down to the first character being 'e' and the final character being 't'. The test would now find the error within 10-30 seconds, which was a great improvement.