NAME: Travis Laxson
ONID: laxsont
DATE: 02-08-2019

Random Testing (RT) - Development Description

The source file testme.c asked that we define the inputChar() and inputString() functions, respectively. The process for rewriting and testing each function is written below.

inputChar() Function:

First, I started with the inputChar() function since, intuitively, this function seemed easier. To develop this function specifically, I began with simple printf statements in order to ensure the function was called successfully. Next, my approach was to identify how the returned character variable was used in the testme() function. I observed that the returned variable from the inputChar() function was set to the character 'c' in the testme() function. Moreover, to figure out which characters the program was testing for 'c' I found the nested if-else statements within the while loop of the testme() function particularly insightful. Once I figured out all of the possible characters that 'c' can be the process of writing the function was straightforward. I initialized a fixed-length character array containing the hardcoded characters in testme() function. Then, using a pseudorandom number generated with the rand() function for a value between zero and nine inclusive I could just return the character at the index of the character array. Lastly, I added another test printf statement to verify the random numbers generated matched the index of the character array.

inputString() Function:

I implemented this function last due to the requirements being a little more challenging. First, I followed the same process as I did with the inputChar() function. This included observing the testme() function closely as to what the inputString() return value should be. Initially, I thought it was a similar return as the inputChar() function, so I began with a similar implementation. However, I noticed that I was only returning the individual characters of the array, namely one of the characters in 'reset'. In other words, I returned a random index value from the hard coded fixed-length array such that the returned character was one of the following: 'r', 'e', 's', 'e', or 't'. After compiling and running this version of the testme program the loop ran infinitely. So I knew the character returned was not enough. Then, I looked a little closer at the nested if statements in lines 62-65. It turns out that I misread these lines initially and instead the returned string should be 'reset' following by a NULL terminatory (\0). Thus, within the function I needed to dynamically allocate some memory since we're returning a character pointer. The hard coded array did not change since I was able to successfully call each character without issue. Next, I realized that because I needed to spell out the word 'reset' then I would have to loop over the length of the array, so I could use the string length library function to ensure I iterate over the five characters. I placed the logic for obtaining the random number within the for loop and then set the dynamically allocated pointer value at the i-th index equal to the random character from the hardcoded array. Since the null terminatory was not included in the array I initialized the last i-th index as the null terminatory (\0). Once I returned the pointer to the hard coded array the testme() function ran successfully and generated the correct output within the required time constraints.