Alexander Tir – tira

Radom Testing Quiz

I followed our lecture notes to develop my random tester. First, I began with examining the preconditions. There were not any preconditions to worry about here since our functions were given to us to fill in. Next, I examined the test oracle, which was also given to us. The test oracle in testme.c prints out “error “ when the final test criteria is met. It also prints the current iteration of the tests, the values of c, s, and the current state the test is in.

Then, I examined the test criteria. Our test criteria was to satisfy states 1 through 10. States 1 through 9 involved having the value of c match certain characters. The final state was to have the string s match the string “reset\0”.

When I began writing the random tester, my first step was to look up the ASCII table. Beginning with inputChar(), I noticed that one method I could use was to generate any random character. This would be the broadest spectrum of test cases, but it is also not a necessary range. Using test case minimization, I noticed that the subset of characters needed to satisfy states 1 through 9 were in the range of 32 to 125, which is the space character “ “ to the closing curly bracket character “}”. This would likely ensure that fewer tests would need to be run. From there, writing the implementation was straightforward. I started with the space character “ “ and added a random range of 0 to 93 to it in order to generate the appropriate minimized character test range:

```
//This function returns a character that I choose to be
//between <space> and } which is the range 32 to 125 on the ASCII table.
char inputChar()
{
  char randomChar;
  // 125-32 = 93, so we will add the range 93 to the space character
  randomChar = ' ' + (rand() % 94);
  return randomChar;
}
```

For inputString(), I used a similar method of test case minimization. Rather than generating any random character, I noticed that the subset of characters needed to satisfy the final state consisted of all lowercase letters. Furthermore, I noticed that the subset of lowercase letters could be further reduced to the subset of characters “e” through “t” in order to satisfy the final state. From there, I wrote the implementation, which would add the a random value in the range of 0 to 15 to the character “e” in order to generate the appropriate range.

```
//This function returns a random string consisting of 5 characters in the range
//e to t which is the range 101 to 116 on the ASCII table.
char *inputString()
{
  static char randomString[6];
  for (int i = 0; i < 5; i++)
  {
    //116 - 101 = 15, so we will add the range 15 to the character 'e'
    randomString[i] = 'e' + (rand() % 16);
  }
    return randomString;
}
```

Finally, I checked my test coverage to ensure that I achieved at least 85% branch coverage. The following is my results from gcov:

```
Function 'testme'
Lines executed:100.00% of 23
Branches executed:100.00% of 50
Taken at least once:96.00% of 50
Calls executed:100.00% of 5
```