Amanda Grant (granaman)
CS362 Fall 2019
10/31/2019
Random String quiz

To run:
```
make testme
```

To view coverage:
```
gcov testme.c -b
```

```
> $ gcov testme.c -b
File 'testme.c'
Lines executed:96.97% of 33
Branches executed:100.00% of 52
Taken at least once:96.15% of 52
No calls
testme.c:creating 'testme.c.gcov'
```

Development
The boilerplate code for this assignment runs a while loop that increments the value of a variable named “state” as long as a particular condition is met for each “promotion”.

- On the first iteration, a randomly-picked value for c must be equal to ‘[‘ for state to be increased from 0 to 1.
- Then, on a subsequent iteration, a randomly-picked value for c must be equal to ‘(‘ and state must already be set to 1 for state to be increased from 1 to 2.
- Then, on a subsequent iteration, c must be equal to ‘{‘ and state must already be set to 2 for state to be increased from 2 to 3.
- (this pattern continues)
- Finally, the string s must be equal to ‘reset’ (literally those letters in that order) and state must be set to 9 in order for the program to end.

This all might seem tremendously unlikely, given that c and s are randomly chosen, but I thought it might become more likely by limiting the character sets that c and s draw from.

For c, I looked up an ASCII table and realized that character 32 was the space needed for promoting state from 3 to 4, and character 125 was the } needed for promoting state from 6 to 7, and every other needed character was somewhere in between those two.

In the C language, setting char to a number effectively sets the char to that character. That made it simple to pick a random number between 32 and 125 and set it to the value returned for c.

char randomChar = (rand() % 94) + 32;

When you look at it that way, it seems a lot more likely that over hundreds of iterations c will eventually be the correct value at the right time and result in the state variable getting promoted all the way from 0 to 9.

Once state is set to 9, then we look at what the s string is set to. This one's a bit trickier because it's looking for a string of a very specific length (6 indices total, the last one must be \0) and all of the previous indices spell 'reset'.

Originally I thought I'd be clever and use the inputChar() method to generate random letters for each position of the string but then I realized that was overkill, and I could instead pick from a more limited set of letters. Hence my letters[10] array that includes e, r, s, t, a bunch of a's and z. The a's are just there as filler. The z is there so I could manually verify that all letters from my letters array were being picked (that is, if I never saw 'z' come up I would know I had an off-by-one somewhere).

Even though it seems rather unlikely that reset would be stepped all the way up to 9 and then the word reset would be randomly picked, it always happens eventually. The amount of runs it takes varies widely. The lowest I saw was 2600 iterations total, and the highest I saw was 382k iterations.

209k iterations
4500 iterations
315k iterations
38k iterations
382k iterations
76k iterations
101k iterations
11k iterations
122k iterations
19k iterations
2600 iterations

On my 2013-era Macbook even the biggest run completes in seconds, so I feel that this is a good size pool of random characters to pick from: small enough such that the code runs in under (well under) the 5 minute limit and big enough that it's not always instantly finished.