Jacob Karcz
7.18.2017
CS-362: SEII
Quiz 2

Quiz 2: Random Testing

The development of the random tester in testme.c went through a few revisions as I slowly realized how much computing time a truly random generator would require.

Since main seeds rand() with srand(time(NULL)) it was apparent from the beginning that c's rand() function would suffice for generating our random characters. This made starting with inputChar() a very simple coding procedure.

Populating the inpuctChar() function was simple, from the get-go I knew I only needed to return one of the 128 possible ascii characters. Initially I thought about getting a random integer in the required range and casting it as a char and then returning that char, but then I remembered we can set a char with its corresponding number and this function therefore needed only a single line of code: return rand() % 127. Because even with all of those possibilities it didn't take long for the tester to enter state 9, I allowed I chose to continue to include all 128 ascii characters, including the 32 unprintable ones. This makes the function more portable and flexible if it needs to be used for another purpose. The calling function can always control for undesired ascii characters.

At first I figured inputString() should return all sorts of random strings containing any of the 128 ascii chars. I thought that a 100 char string was a decent length for the maximum possible length. I revised this because the longest word in the English language is "only" 45 characters. Fortunately I decided to first test my function using only 5 character strings, since this would allow me to target my test towards the "reset" string we're looking for. I started out by using 5 character strings with any of the 128 ascii chars. After reading some of the posts on canvas, I realized this assignment didn't have to be so random. Going over the lecture materials confirmed that, since we do want to target our tests towards our test cases (though less so with these random testers). So first I decided to use 5 character strings containing any of the ~97 printable ASCII chars. This was still taking a very long time on flip (it never did complete). So I decided to only do A-z. This was still taking a long time to hit the coveted "reset" combination, and after more reading through Piazza I decided it was best to just use lowercase ascii characters to populate the 5 character string. This returns in an acceptable "few minutes." The longest time it took to print the error message and exit with code 200 was about 18 million iterations but often it completes this in 1-5 million iterations. Which isn't too bad.

Because I still like the "enhanced" randomness I conceived earlier, I left them as precompiler options in my testme.c file:

```
#define MAX_LENGTH 5
#define FIXED 0 //0 == fixed @ MAX_LENGTH, 1 == random string length up to MAX
#define RNDMNESS 0 //0 == lowercase ascii, 1 == A-z, 2 == printable ascii, 3 == all ascii
```

Which correspond with:

```
if (FIXED) length = (rand() % MAX_LENGTH) + 1; //1-MAX chars
else length = MAX_LENGTH;
…
if (RNDMNESS == 3) x = inputChar();  // all ascii
if (RNDMNESS == 2) x = (rand() % (123 - 32)) + 32; // printable ascii
if (RNDMNESS == 1) x = (rand() % (123 - 65)) + 65; // A-z
if (RNDMNESS == 0) x = (rand() % (123 - 97)) + 97; // lowercase ascii
```

So the default for a speedy test should be as above, where MAX_LENGTH is 5 (any less and it will never hit "reset"! FIXED is 0 so that the string returned by inputString() is MAX_LENGTH long (5 chars) and RNDMNESS at 0 so that we only use lowercase letters and speed up the process.

We can easily increase the randomness by changing these settings, however since the objective of our random tester is to come up with the string "reset," these options should be enough. If we want to be more thorough (and dedicate more time and memory resources) we can increase the randomness by allowing other ascii chars and strings of varying length. These options also allow for increased protablity and flexibility fon the inputString() function, if we're looking for strings of different length or that include other ascii characters and not just lowercase letters.

I know we weren't supposed to change anything in testme() but the memory leaks were so bad that even the flip server was killing my program early. I added a couple of lines to free up the discarded string pointers, since good memory management is an important part of any program. Arguably, it would be more important in a testing program since it could lead to a never-ending bug-hunt to plug a non-existent leak in the program we're testing! Although valgrind would quickly tell the tested exactly where the leak is coming from. Regardless, I plugged the leak because it was consuming gigabytes of memory (one of the terminations had it at 6.2GB).

The way my solution works is that inputChar() first returns any of the 128 possible ascii chars (since it hits state 9 quickly, I preferred to leave the non-printable chars as well, as I mentioned earlier). Then the random string is generated, depending on the precompile options set (as I described it). This can fluctuate the level of randomness, but as it is set now (above) to minimize the time required while still generating completely random string, the tester returns lowercase letter strings that are 5 characters long. Eventually, usually within the first 5 million iterations, we get the "reset" string.

Overall, I think my testing was fairly successful and although my program is not as random as it could be, given memory and time constraints I think I made the right choices in budgeting my resources while retaining the possibility to implement my initial testing ideas with increased randomness (probably for testing something else).