Isaac Neibaur

CS362 Summer Quiz 1

7/24/2019

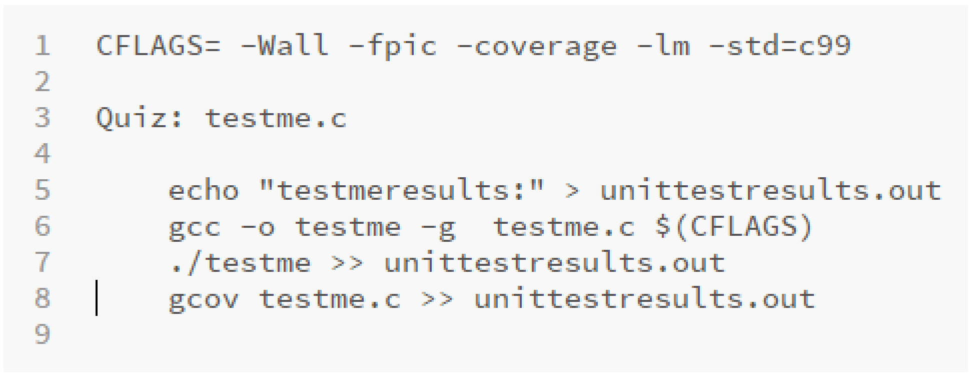
Documentation:

a. Describe in detail how you developed your random tester in a file called randomstring.pdf.

In completing this assignment at first I visually inspected the code. Of note is the "while (1)" loop. Inside the while loop is an exit line, however it will only be triggered if the state is equal to 9 AND the string in question is "reset" ending in a null pointer. The state is only equal to 9 if a specific state and single character combination are met. Fortunately, once the state is 9, it will not change to any other value.

Therefore, to limit the range of variables, for the characters in question I limited the scope to only characters that could change the state variable. With my string function, I limited the characters in question to "rest" and made sure all strings returned with a length of 5 and with the final character being a "0". This should increase the likelihood that a random combination will eventually be "reset" when the state is 9.

Note, prior to implementing this code, I first created a makefile. Similar to prior assignments I pushed output to a text file.



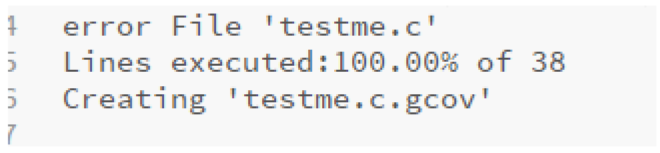
```
CFLAGS= -Wall -fpic -coverage -lm -std=c99

Quiz: testme.c

    echo "testmeresults:" > unittestresults.out
    gcc -o testme -g  testme.c $(CFLAGS)
    ./testme >> unittestresults.out
    gcov testme.c >> unittestresults.out

```

However, in order to get my output text file to include the lines executed I had to introduce a variable so I could effectively end the while loop instead of error out of the program when all conditions were met. If so, at the end of my text file I saw something similar to below. If I left the exit call in my code, the below did not print, instead I only saw at the end of the printout the "error" message



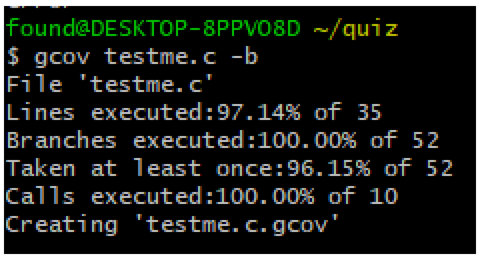
```
error File 'testme.c'
Lines executed:100.00% of 38
Creating 'testme.c.gcov'

```

With the very first testing, I tested to make sure the makefile worked, without implementing any new code in the testme.c file. It was quickly apparent that this was a mistake, as the infinite loop was expected, however my text output file had ballooned to an enormous size, over 2gb! With the initial coding, I tried at first just to code the basic character and string to trigger the exit statement. To simplify this testing, I simply set state to 9 and then focused on having my inputString() function just return "rest\0". Once this was working as expected I implemented a random feature to both functions and again tested to make sure code could still compile and run as expected.

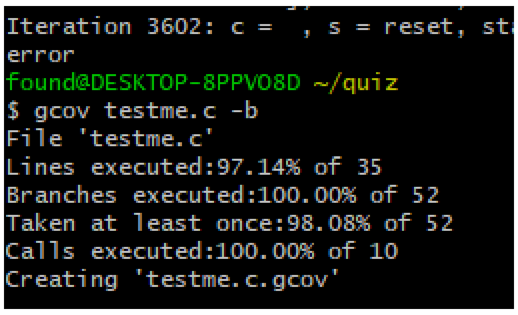
I did search for ways to implement random strings in C and found the following to be a helpful resource https://www.codeproject.com/Questions/640193/Random-string-in-language-C. There they were looking at all alpha characters and were generating random strings. However, the code was easily modified to fit the needs of this quiz with both a random string an random character generator.

After running the code I exectuted the testme.c function. Reviewing the the coverage with "gcov testme.c -b", shows that this test does have great branch coverage (100%)



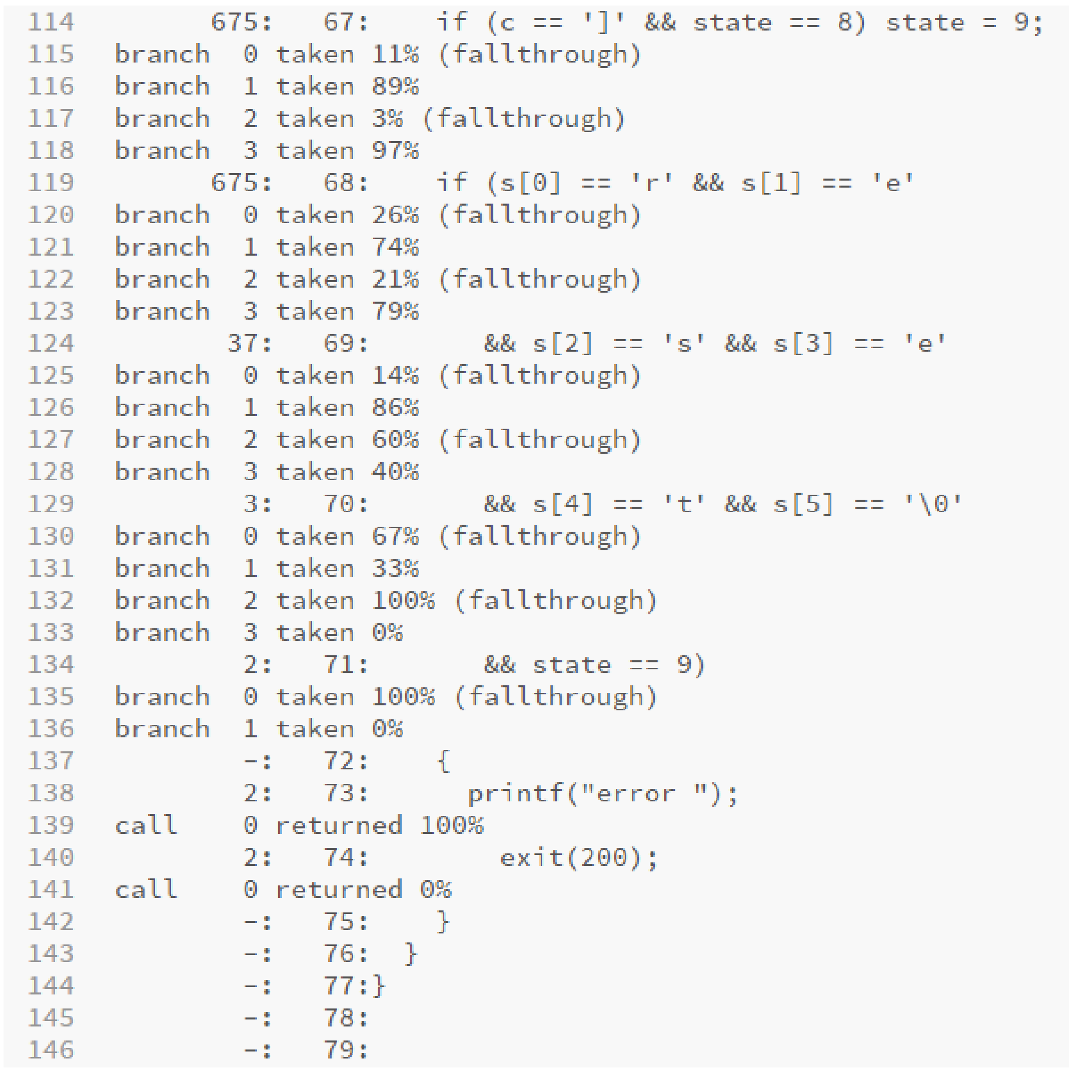
```
found@DESKTOP-8PPVO8D ~/quiz
$ gcov testme.c -b
File 'testme.c'
Lines executed:97.14% of 35
Branches executed:100.00% of 52
Taken at least once:96.15% of 52
Calls executed:100.00% of 10
Creating 'testme.c.gcov'
```

Note, multiple runs yielded very similar data



```
Iteration 3602: c =  , s = reset, st
error
found@DESKTOP-8PPVO8D ~/quiz
$ gcov testme.c -b
File 'testme.c'
Lines executed:97.14% of 35
Branches executed:100.00% of 52
Taken at least once:98.08% of 52
Calls executed:100.00% of 10
Creating 'testme.c.gcov'
```

Further running the testme executable and then opening up the gcov file does show the hits per each line of the program and considering the code at hand these coverages make sense.



```
      675:   67:    if (c == ']' && state == 8) state = 9;
branch  0 taken 11% (fallthrough)
branch  1 taken 89%
branch  2 taken 3% (fallthrough)
branch  3 taken 97%
      675:   68:    if (s[0] == 'r' && s[1] == 'e'
branch  0 taken 26% (fallthrough)
branch  1 taken 74%
branch  2 taken 21% (fallthrough)
branch  3 taken 79%
       37:   69:       && s[2] == 's' && s[3] == 'e'
branch  0 taken 14% (fallthrough)
branch  1 taken 86%
branch  2 taken 60% (fallthrough)
branch  3 taken 40%
        3:   70:       && s[4] == 't' && s[5] == '\0'
branch  0 taken 67% (fallthrough)
branch  1 taken 33%
branch  2 taken 100% (fallthrough)
branch  3 taken 0%
        2:   71:       && state == 9)
branch  0 taken 100% (fallthrough)
branch  1 taken 0%
        -:   72:    {
        2:   73:      printf("error ");
call    0 returned 100%
        2:   74:        exit(200);
call    0 returned 0%
        -:   75:    }
        -:   76:  }
        -:   77:}
        -:   78:
        -:   79:
```