Kirsten Wollam, wollamk

CS 362 Spring 2019

Random Test Quiz

To develop my random tester for the testme() function I first started by examining the function to determine what it was doing and what inputs would need to be tested in the random tests. I also considered what would be able to run in a reasonable time and get full code coverage. The meat of the function is in this while loop:

while (1){

tcCount++;

c = inputChar();

s = inputString();

printf("Iteration %d: c = %c, s = %s, state = %d\n", tcCount, c, s, state);

if (c == '[' && state == 0) state = 1;

if (c == '(' && state == 1) state = 2;

if (c == '{' && state == 2) state = 3;

if (c == ' '&& state == 3) state = 4;

if (c == 'a' && state == 4) state = 5;

if (c == 'x' && state == 5) state = 6;

if (c == '}' && state == 6) state = 7;

if (c == ')' && state == 7) state = 8;

if (c == ']' && state == 8) state = 9;

if (s[0] == 'r' && s[1] == 'e'

&& s[2] == 's' && s[3] == 'e'

&& s[4] == 't' && s[5] == '\0'

&& state == 9)

{

printf("error ");

exit(200);

From this I first listed out the minimum necessary characters that c would need to be able to be: [({ ax})]

Similarly s would need to be able to at a minimum contain the characters: rest

If the random generation functions did not contain these as possibilities there would be no way to every get out of the while loop and get 100% coverage.

For the inputChar function I decided on the following random generator:

char inputChar()

{

char c = "[({ abcdefghijklmnopqrstuvwxyz})]"[rand() % 33];

return c;

}

This will choose a character at random from the string [({ abcdefghijklmnopqrstuvwxyz})] and then return it. I included all the lowercase letters as well as the other required characters. I chose this as the requirements for c are pretty easy to meet and adding more characters would give it more tests on those lines as it made its way through the states.

For inputString I used the following random generator:

char \*inputString()

{

char \*s = malloc(sizeof(char) \* (6));

int i;

for( i = 0; i < 5; i++){

s[i] = "reset"[rand() % 5];

}

return s;

}

This allocates an array and fills it with 5 characters randomly from the set: reset

I chose to limit the random options in this case to just those that the string is looking for because they must be found is a specific order to meet the criteria of that if statement. I did not want the test to take too long to run, so I chose something that would still take many iterations, but not so many that the program would take several minutes to run.

As expected these tests run reasonably quickly, taking between 200 and 4000 iterations on the while loop in my trials, and provide 100% code coverage.

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'

The only line of code not executed is the final return, as the program is designed to error out before it can be reached.