Class: CS-362
Term: Summer 2017
Author: Jon-Eric Cook
Date: July 16, 2017
Quiz: #2

Development

The beginning of this random tester started with spending 10-15 minutes stepping through the testme function to get an idea of what was going on. Essentially, when certain character and integer combinations (c and state) are present, the program is able incrementally advance through a series of nine if statements. Once the ninth if statement is true, the last if statement checks if a random string contains a specific sequence of characters. Once the random string matches the specific sequence of characters, the error is found and the program exits. See code below for clarity.

```
while (1) {
   tcCount++;
   inputChar(cPtr);
   inputString(s);
   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);
   }
}
```

Once the above was firmly understood, the next step was to determine how to utilize the two supporting functions inputChar and inputString and what range of characters from the ASCII table should be used in the test. inputChar sets the value of character “c” (see code above) to something random, ranging from ‘ ‘ (Space) to ‘~’. On the ASCII table these values are 32 and 126 respectively. inputString sets the first 5 characters of string “s” (see code above) of length 6 (the last character is ‘\0’) to something random, ranging from ‘ a‘ to ‘z’. On the ASCII table these values are 97 and 122 respectively. The testme function is tested by calling the above two functions repeatedly until inputChar returns all nine of the needed characters to pass all the if statements and inputString randomly generates the string “reset”.

The character range of ‘Space’ to ‘~’ was chosen for inputChar because it was the entire ASCII table of appropriate characters that could be tested. With the processing power available today, returning a random character from a pool of 95 possible characters was trivial. The range of ‘a’ to ‘z’ was chosen for inputString because it was the entire lowercase alphabet and only lowercase letters were tested. Again, with the processing power available today, generating a string of five random characters from a pool of 26 was trivial. See code below for clarity.

```
void inputChar(char *character) {
    // selecting random characters from 'Space' to '~' on the ASCII table
    int min = 32; // 'Space'
    int max = 126; // '~'
    int delta = max - min;
    *character = (rand() % (delta + 1)) + min;
}

void inputString(char string[]) {
    // selecting random characters from 'a' to 'z' on the ASCII table
    int min = 97; // 'a'
    int max = 122; // 'z'
    int delta = max - min;
    int i;
    for (i = 0; i < 5; i++) {
        string[i] = (rand() % (delta + 1)) + min;
    }
}
```

The above ranges were not chosen first. The first choice of characters for inputChar was simply “[({ ax})]”, all the characters that would pass the nine if statements. The first choice of characters for inputString was “reset”, the string that would pass the last if statement. With these two choices of possible characters, the test ran very fast, at times taking less than 200 iterations. It was discovered though that by only using the above characters, no “stray” character or “random” character from the ASCII table could be tested. Upon this realization, both the ranges for inputChar and inputString were widened to the entire ASCII table, ranging from ‘Space’ to ‘~’.

By drastically increasing the character range, the program ran for tens of millions of iterations, mostly due to the random string generation having such a large pool of characters to choose from. From this, it was determined that this range for inputString was too large, as the time to completion would be too long. At this point, the range for inputString was settled on to be from ‘a’ to ‘z’. Even though the character range was cut down from the entire ASCII table to only the lowercase alphabet, there was still a plethora of options to generate a random string that tests the limits of the testme function and has a reasonable time to completion.