**Testme()**

The testme() function is a loop that calls inputChar() and inputString() functions in each iteration, until both of the following conditions are met:

1. The integer variable ‘state’ == 9.
2. The string array variable ‘s’ equals [r, e, s, e, t, \0].

Important variables in the testme() function:

1. The char\* variable 's' holds the result of inputString().
2. The char variable 'c' holds the result of inputChar().
3. The integer variables 'state' and 'tcCount' are initialized to 0.
   1. The 'tcCount' variable tracks the number of iterations.
   2. The 'state' variable is incremented when a character of interest is encountered, in the predetermined order listed below in the function.

**inputChar()**

Calling the inputChar() function will select a random character from the ASCII character set. The characters of interest are within the ASCII range of 32 - 125 inclusive. Thus, the inputChar function should be capable of producing a character of interest ('[', '(', '{', ' ', 'a', 'x', '}', ')', ']') upon each call. In order to achieve this desired output, I wrote the inputChar() function as follows:

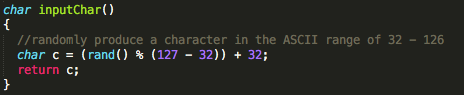


Figure . inputChar() function

**inputString()**

Calling the inputString() function generates a randome string that is six characters long. The 'string of interest' for the testMe() method is "reset\0", so inputChar()must be capable of producing this array of characters. In order to increase the chances of producing "reset\0", the following have been implemented:

1. The ASCII range from which random characters may be selected is limited to 101 - 116 (e - t), inclusive.
2. The last character in the string of 6 is hard-coded to '\0'.

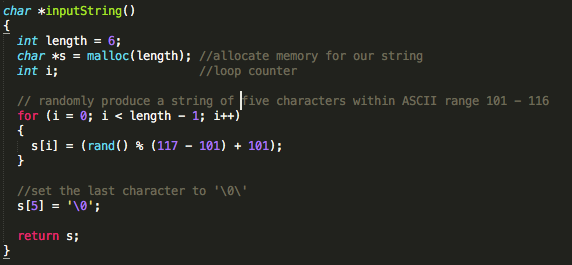


Figure . inputString function

**Testing Notes:**

1. I commented out the printf line that was produced in each iteration of the while(1) loop inside the testme() function. The loop ran a minimum of ~100,000 times during each test, and generating a print statement for each iteration was inefficient and very distracting.
2. I moved the same print statement to the end of the while loop, so that it would be executed just before the function exited (Figure 3). This way, the printf statement only executes once for each test, and gives the programmer important information, such as the number of iterations to error/exit, as well as the ending character, string, and state.
3. Some testing results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Iterations** | **‘c’** | **‘s’** | **‘state’** |
| 561746 | m | reset | 9 |
| 319373 | t | reset | 9 |
| 1751592 | f | reset | 9 |
| 1396914 | Q | reset | 9 |
| 3447292 | b | reset | 9 |
| 103218 | . | reset | 9 |

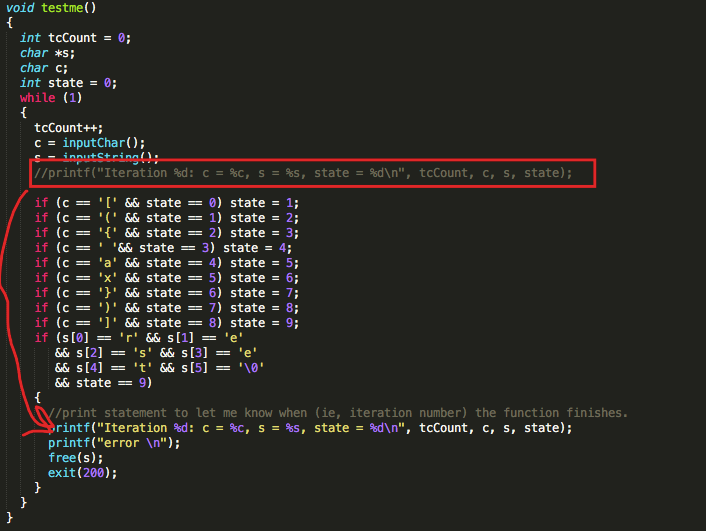


Figure , printf statement moved to the end of the while() loop.