Test cases for the function: int findString(const char str[], const char list[][MAX\_STRING\_LEN + 1], const int nstrings);

|  |  |  |  |
| --- | --- | --- | --- |
| **Function: findString** | **Data** | **Expected Result** | **Description** |
| Test 1 (black box) | str="abc" list = {"abc", "def", "ghi"}  nstrings=3 | 0 | Tests to see if a string is found in the first position of string array. |
| Test 2 (black box) | str="def" list = {"abc", "def", "ghi"}  nstrings=3 | 1 | Tests to see if a string is found in the second position of string array. |
| Test 3 (black box) | str="xyz" list = {"abc", "def", "ghi"}  nstrings=3 | -1 | Tests to see if a string is not found in the string list array. |
| Test 4 (white box) | str="ghi" list = {"abc", "def", "ghi"}  nstrings=3 | 2 | Tests to see if a string is found in the third position of string array. |
| Test 5 (white box) | str="seneca" list = {"abc", "def", "ghi"}  nstrings=3 | -1 | Tests to see if a string is not found in the string list array. |
| Test 6 (white box) | str="abc" list = {}  nstrings=0 | -1 | Tests to see if a string is found but the list of strings is empty. |

Test cases for the function: void init(int ar[], const int value, const int size);

|  |  |  |  |
| --- | --- | --- | --- |
| **Function: init** | **Data** | **Expected Result** | **Description** |
| Test 1 (black box) | ar = {0, 0, 0, 0}  value = 5  size = 4 | ar = {5, 5, 5, 5} | Initialize all members of an existing array of size 4 to a single value of 5. |
| Test 2 (black box) | ar = {1, 2, 3, 4}  value = 7 size = 4 | ar = {7, 7, 7, 7} | Initialize all members of an existing array of size 4 to a single value of 7. |
| Test 3 (black box) | ar = {}  value = 6 size = 0 | ar = {} | Initializing of an array of size 0 lead to no change in the array. |
| Test 4 (white box) | ar = {1, 2, 3, 4, 5}  value = 0 size = 5 | ar = {0, 0, 0, 0, 0} | Initialize all members of an existing array of size 5 to a single value of 0. |
| Test 5 (white box) | ar = {4, 3, 2, 1}  value = 4 size = 4 | ar = {4, 4, 4, 4} | Initialize all members of an existing array of size 5 to a single value of 0. |
| Test 6 (white box) | ar = {}  value = 3 size = 0 | ar = {} | Initializing of an array of size 0 lead to no change in the array. |

Test cases for the function: int add2Cart(struct Cart\* cart, const int item);

|  |  |  |  |
| --- | --- | --- | --- |
| **Function: add2Cart** | **Data** | **Expected Result** | **Description** |
| Test 1 (black box) | cart = {items: [], num\_items: 0}  item = 2 | 0 | Adds an item = 2 to the struct cart which was empty before. |
| Test 2 (black box) | cart = {items: [1, 2, 3], num\_items: 3}  item = 2 | 0 | Adds an item = 2 to the struct cart which contained 1, 2 and 3 in it before. |
| Test 3 (black box) | cart = {items: [3, 2, 1], num\_items: 3}  item = 4 | 0 | Adds an item = 4 to the struct cart which contained 3, 2 and 1 in it before. |
| Test 4 (white box) | cart = {items: [1, 3, 5], num\_items: 3}  item = 4 | 0 | Adds an item = 4 to the struct cart which contained 1, 3 and 5 in it before. |
| Test 5 (white box) | cart = {items: [1, 3, 5], num\_items: 11} item = 4 | -1 | Shows -1 because 11 excedes the macro MAX\_CART for max number of items in a cart. |
| Test 6 (white box) | cart = {items: [1, 3, 5], num\_items: 3} item = 11 | -2 | Shows -2 because 11 excedes the macro MAX\_PRODUCTS for max number of products. |

**Reflection:**

1. I was able to find more test cases by using the white box method. In white box method, the tester is able to view the code while testing which can help the code tester in finding the edge cases and testing them, which can help reduce bugs. No, I believe that not adequate testing can be done with just one of the techniques because if we want to test from a user’s perspective, we have to use the black box testing method and if we want to test from the programmer’s perspective, we have to use white box testing method. Black box tests were easier to develop because I just needed to enter some input values and check the output without caring about the code written in the program. I think black box testing method is faster than white box testing method because in white box testing additional time is spent for checking the code and creating test cases accordingly.
2. For the above given function, I would follow a systematic approach where I will test that both the functions work together seamlessly and give the correct output. I would first try real world normal test cases which ensures that the program runs nicely in best possible conditions. Then, I would try edge cases to find out that the program also works fine in the border conditions and does not give a bug as the output. Yes, I would need additional code to set up and run the test because I will need to combine the functions and make sure that they work in unity and by looking at the test cases created for that particular code, I could compare and ensure that the program containing the integration of both the functions works fine. For writing the additional code, I think I would need around 45 mins to 1 hr 30 mins of extra time.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#define MAX\_STRING\_LEN 50

struct Cart {

int items[10];

int num\_items;

};

int findString(const char str[], const char list[][MAX\_STRING\_LEN + 1], const int nstrings) {

for (int i = 0; i < nstrings; i++) {

if (strcmp(str, list[i]) == 0) {

return i;

}

}

return -1;

}

int add2Cart(struct Cart\* cart, const int item) {

if (cart->num\_items >= 10) {

return -1;

}

cart->items[cart->num\_items] = item;

cart->num\_items++;

return 0;

}

int main() {

// set up

struct Cart cart = {{1, 3, 5}, 3};

const char carts[][MAX\_STRING\_LEN + 1] = {"flour", "sugar", "bananas", "potatoes"};

const int num\_items = 4;

const char items[] = "cherry";

// execution

int result = add2Cart(&cart, findString(items, carts, num\_items));

// comparison

if (result == 0) {

printf("The items have been successfully added to the cart.\n");

} else {

printf("Sorry, items have not been added to the cart.\n");

}

return 0;

}