|  |  |  |  |
| --- | --- | --- | --- |
| Function: findString | Data | Expected Result | Description |
| Test 1 | str = "abc", list = { "abc", "def", "ghi" }, nstrings = 3 | 0 | Tests if the string "abc" is found in the first position of the string array. |
| Test 2 | str = "def", list = { "abc", "def", "ghi" }, nstrings = 3 | 1 | Tests if the string "def" is found in the second position of the string array. |
| Test 3 | str= “123”, list = { "123", "456", "789" }, nstrings = 3 | 2 | Tests if the number string works well and if string is found in the second position of string array. |
| Test4 | str=" ", list={"abc", "def", "ghi", "jkl"}, nstrings=4 | -1 | Test if the string is EMPTY return’s False or not. |

**Black Box Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Function: init | Data | Expected Result | Description |
| Test 1 | ar = [1, 2, 3, 4, 5],value=0,size=5 | ar = [0, 0, 0, 0, 0] | Tests if all members of the array 'ar' are initialized to the value 0. |
| Test 2 | ar = {0, 0, 0, 0, 0},value=0,size=5 | No expected output | Initialize array of size 5 with value 0 |
| Test 3 | ar = {-1, -1, -1, -1, -1, -1, -1, -1},value=-1,size=8 | No expected output | Initialize array of size 8 with value -1 |
| Test 4 | ar: {},value=5,size=0 | No expected output | Initialize empty array of size 0 with value 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| Function: add2cart | Data | Expected Result | Description |
| Test 1 | cart = { items: [0, 1, 2], nItems: 3 }  ,items=3 | 0 | Tests if the item 3 is successfully added to the cart. |
| Test 2 | cart = { items: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], nItems: 10 }, items=10 | -1 | Tests if an error occurs when adding an item to a full cart. |
| Test 3 | cart = { items: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], nItems: 10 },items=-1 | -2 | Tests if an error occurs when trying to add an invalid item to the cart. |
| Test 4 | cart={ } , item=-1 | Non zero value | Tests if negative items returns false |

|  |  |  |  |
| --- | --- | --- | --- |
| Function: clear | Data | Expected Result | Description |
| Test 1 | input= "!@#$%^&\*()\_+\n" | No output expected | Clear input buffer with spaces and special characters |
| Test 2 | input=NULL | No output expected | To check if it passes NULL or not |
| Test 3 | input="\n\n\n\n" | No output expected | Clear input buffer with multiple newline characters |
| Test 4 | input=" " | No output expected | To check if its passes empty value |

**White Box Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Function: init | Data | Expected Result | Description |
| Test 1 | ar = [], value=1, size=5 | ar = [] | Tests if the function correctly handles an empty array. |
| Test 2 | input=[5, 10, 15, 20], value=-7, size=4 | No output expected | Initialize array with negative value |
| Test 3 | input=[0, 0, 0, 0, 0], value=0, size=5 | No output expected | Initialize array with zero value |
| Test 4 | input=[1,2,3], value=100, size=5 | The input array should be initialized with all elements set to 100. | Initialize array with value greater than maximum |

|  |  |  |  |
| --- | --- | --- | --- |
| Function: findString | Data | Expected Result | Description |
| Test 1 | str = "", list = { "abc", "def", "ghi" },  nstrings=3 | -1 | Tests if an empty string is handled correctly. |
| Test 2 | str = "abc", list = { "abc", "def", "abc" }, nstrings=3 | 3 | Tests if the function returns the index of the first occurrence when a string is present multiple times in the list. |
| Test 3 | str=NULL, list={"abc", "abc", "abc"}, nstrings=3 | No output expected | To check if it passes NULL or not |
| Test 4 | str= “ ”, list={"abc", "abc", "abc"}, nstrings=4 | No output expected | The function should return the position of the first occurrence of "abc" in a list, which is 0 because all the strings in the list are the same. |

|  |  |  |  |
| --- | --- | --- | --- |
| Function: add2cart | Data | Expected Result | Description |
| Test 1 | cart = { items: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],nItems=3 },items=3 | -1 | Tests if an error occurs when adding an item to a full cart. |
| Test 2 | cart = { items: [empty cart], nItems= 0 }, items=3 | -2 | Add item to empty cart |
| Test 3 | cart={items[MAX\_CART], nitems=6 }, item=5 | Program will not run | Tests if an error occurs when trying to add an invalid item to the cart. |
| Test 4 | cart={ } , nitems=0,item=-1 | Program will not run | To see unexpected results |

|  |  |  |  |
| --- | --- | --- | --- |
| Function: clear | Data | Expected Result | Description |
| Test 1 | input=" " | No output expected | Clear empty input buffer |
| Test 2 | Input= “Line 1\nLine 2\nLine 3\n” | The function should clear the input buffer until the last newline character '\n' is reached. | Clear input buffer with multiple lines |
| Test 3 | input=NULL | No output expected | To check if it passes NULL or not |
| Test 4 | Input= “!@#$%^&\*()\_-+={[}]|\:;"'<,>.?/\n” | No output expected | Clear input buffer with special characters |

Reflection

* The initial series of test cases were identified via black field testing, which attempted to evaluate the performance of the functions that met with their specifications without analysing the code implementation. Then, using white box testing, I created additional test cases by inspecting the code and deliberating on its underlying logic and paths. Because black field testing does not include diving into the inner workings of the code, it is often much faster to execute. They can be completed more quickly because black field checks can be established and completed using only the characteristic's specification. White box testing, compared to black box testing, which might take more time, comprises reviewing the code, discovering paths, and implementing checks based entirely on internal behaviour. Overall, a mix of black box and white box testing ensures a higher level of courage in the software's satisfactory by verifying both internal code paths and practical demands.
* To develop integration cases, test scenarios that model the interaction of various features must be created. The test environment must be configured, and the tests must be executed, which requires more code. Code examples include initializing data structures, calling functions in a specific order, and getting output. Make the connection between the two functions, findString and add2Cart, within the code. We can maintain the function code the same while avoiding changing variable values and other issues. This requires comparing the output or the order produced by the characteristic calls to what was predicted. Writing integration tests often requires more effort and time than writing unit tests because to they require to manage multiple elements and explore their relationships.
* Set up

char descriptions[][MAX\_STRING\_LEN + 1] = {"apple", "banana", "orange", "grape", "mango"};

int numStrings = MAX\_PRODUCTS;

struct Cart { int items[MAX\_PRODUCTS]; int nItems; } cart = {{0}, 0};

init(cart.items, -1, MAX\_PRODUCTS);

Execution and Comparison:

int index = findString("orange", descriptions, numStrings);

int addResult = add2Cart(&cart, index);

if (addResult == 0 && cart.items[0] == index && cart.nItems == 1) printf("Integration Test Passed\n");

else printf("Integration Test Failed\n");

return 0;