SFT211 – Reflection on weekly workshop2.

I found more test cases via black box testing than white box testing. I don’t believe that adequate testing could be done with just one of the techniques. In my development for the test cases, both of these techniques complemented each other. Black box testing allowed me to get a general idea of what I would get as an output for an input by running the software. And white box testing allowed me to check the actual code, what is being done to get what I am getting. Combining both of them ensures thorough testing of both functionality and code structure. Black box testing ensures that the software meets the user's requirements, while white box testing ensures that the code is robust and error-free. For this specific workshop, developing black box tests was easier. I could put in inputs and based on my expected output, I would compare the actual output. This allowed me to develop more cases than white box testing did. Because when I investigated the code, it limited my horizon as to what type and how many inputs are eligible for the software. And developing black box was faster than white box again for the same reason. In black box I put in input and examined output, whereas in white box I had to thoroughly examine the code, look at the conditions and then to test it, which took some time.

Integration testing involves verifying that different modules or functions interact correctly and produce the expected outcomes when used together. To set up integration tests for the provided functions, I will first have to identify scenarios where multiple functions interact. Then I will create realistic use cases that involve sequences of function calls. Prepare necessary data structures and inputs and ensure that the environment mimics the real-world usage as closely as possible. I will need to create additional code to set up and run the tests. As the provided functions are only definitions with no codes in them. I will write the code to compare actual outputs against expected results. One of the integration test could be- “The shopping cart scenario” – initialize the cart, add items to the cart using “add2Cart”, verify items are added correctly, use “findString” to verify product lookups, and lastly use “clear” function to reset the test environment, ensuring no side effects of the subsequent tests. Writing the additional code for setting up and running integration tests will require careful planning and design. Implementing the helper functions, test cases, and verifying the outputs might take several hours to a few days for me, depending on the complexity of the functions and the thoroughness of the tests. Debugging and refining the tests to ensure they cover all necessary interactions also adds to the time required.

The 2 functions that I used for integration are ‘findString’ and ‘add2Cart’. The former used to find a position of an item in the product descriptions while the latter used to add the found item to the cart. And verify visually that they have been added correctly. Here is the code:

#include <stdio.h>

#include <string.h>

#define MAX\_STRING\_LEN 30

#define MAX\_PRODUCTS 10

#define MAX\_CART 10

struct Cart {

int items[MAX\_CART];

int nItems;

};

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

void init(int ar[], const int value, const int size);

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

void clear();

// Helper function to compare arrays

int compareArrays(int arr1[], int arr2[], int size) {

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

if (arr1[i] != arr2[i]) {

return 0; // Arrays are not equal

}

}

return 1; // Arrays are equal

}

**// Integration test combining add2Cart and findString**

void testAdd2CartFindStringIntegration() {

// Setup part of the program

struct Cart cart;

char descriptions[][MAX\_STRING\_LEN + 1] = {

"flour", "sugar", "bananas", "potatoes", "milk", "tea"

};

int numProducts = 6;

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

cart.nItems = 0;

// Execute part of the program

// Find the position of "milk" in the descriptions

int pos = findString("milk", descriptions, numProducts);

// Add the found item to the cart

int rc = add2Cart(&cart, pos);

// Comparison

int expectedItems[MAX\_CART] = {4, -1, -1, -1, -1, -1, -1, -1, -1, -1};

if (rc == 0 && cart.nItems == 1 && compareArrays(cart.items, expectedItems, MAX\_CART)) {

printf("Integration test (add2Cart + findString): PASSED\n");

} else {

printf("Integration test (add2Cart + findString): FAILED\n");

}

}

int main(void) {

testAdd2CartFindStringIntegration();

return 0;

}

// Function implementations

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

int i, result = -1, found = 0;

for (i = 0; i < nstrings && !found; i++) {

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

result = i;

found = 1;

}

}

return result;

}

void init(int ar[], const int value, const int size) {

int i;

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

ar[i] = value;

}

}

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

int result = 0;

if (cart->nItems >= MAX\_CART) {

result = -1;

} else {

if (item >= 0 && item < MAX\_PRODUCTS) {

cart->items[cart->nItems] = item;

cart->nItems++;

} else {

result = -2;

}

}

return result;

}

void clear() {

char ch;

while ((ch = getchar()) != '\n');

}

--The program successfully recognizes ‘milk’ from the list of descriptions of items and prints, **Integration test (add2Cart + findString): PASSED.**

**Setup**: Initializing the cart using the init function. Preparing the product descriptions and setting the initial number of products.

**Execution**: Using the findString function to find the position of the product "milk". Using the add2Cart function to add the found item to the cart.

**Comparison**: Verifying the return code of add2Cart. Verifying the number of items in the cart (cart.nItems). Comparing the cart's items array to the expected result using the compareArrays helper function.

Adding more test cases to this integration should be obvious, just change the arguments in the findString function.