SFT 221

Workshop 2

Name:	patel vrundaben vijaykumar
ID:	158605220
Email:	vvpatel20@myseneca.ca
Section:	ngg

Authenticity Declaration:
I declare this submission is the result of my own work and has not been shared with any other student or 3rd party content provider. This submitted piece of work is entirely of my own creation.

FUNCTION: findString						
Test #	Test Type	PARAMETERS			Expected Results	Description
		str	list	nstrings		
1	Blackbox	str="flour"	list="flour" "sugar" "bananas" "potatoes" "milk" "tea"	6	0	This test checks if the function correctly finds the position of a string ("flour") in an array of strings. The input string ("flour") exists in the list, and the expected result is the index position 0.
2	blackbox	str="soap"	list="flour" "sugar" "bananas" "potatoes" "milk" "tea"	6	-1	This test checks if the function handles the case when the input string ("soap") is not in the list. The expected result is -1, indicating that the string was not found
3	white box	str="bananas"	list="flour" "sugar" "bananas" "potatoes" "milk" "tea"	6	2	This test checks if the function correctly finds the position of a string ("bananas") in an array of strings. The input string ("bananas") exists in the list, and the expected result is the index position 0.
4	white box	str=""	list="flour" "sugar" "bananas" "potatoes" "milk" "tea"	6	-1	this test check if the function corretly handles the case when the input string is empty. The expected result is -1 as the empty string is not found in the list
5	white box	str="tea"	list=""	0	-1	this test check if the function corretly handles the case when the list nstrings is empty. The expected result is -1 as there are no strings in the list
6	white box	str="sugar"	list="flour" "sugar" "bananas" " sugar" "potatoes" "milk" "tea"	7	1	This test case examines the behavior when list contains duplicate strings. The function should return the position of the first occurrence. The expected result is 1 as " sugar" is in the 1st index in the first occurence

FUNCTION: init						
Test #	Test Type	PARAMETERS			Expected Results	Description
		ar	value	size		
1	Blackbox	ar ={ 1, 2, 3, 4, 5}	0	5	{0, 0, 0, 0 , 0}	this test case checks if the function initializes an array of size 5 with the value 0 correctly. The expected result is that all elements in the array are set to 0.
2	blackbox	ar={}	5	0	{}	This test case verifies if the function handles the scenario when attempting to initialize an empty array of size 0 with the value 5. the expected result is that since there are no elements in array, no action should be taken.
3	whitebox	ar={0}	27	1	{27}	This test case verifies if the function sets all elements of an array to the specified value 27 when size is 1. the expected is that ut should initialize the single element in the array.
4	blackbox	ar ={ 1, 2, 3, 4, 5}	-1	5	{-1, -1, -1, -1, -1}	the test case checks if the function initializes an array of size 5 with the value -1 correctly. The expected result is that all elements in the array are set to -1.
5	whitebox	ar ={ 1, 2, 3, 4, 5}	0	2	{0, 0, 3, 4, 5}	the test case checks if the function initializes an array consisted of 5 elements and the value 0 and the size is 2. this would result in undefined behavior, the function initializes only the first two elements of the array, and the remaining elements retain their uninitialized values.

FUNCTION: add2Cart					
Test #	Test Type	PARAMETERS		Expected Results	Description
		cart	item		
1	Blackbox	empty cart	2	0	the test case checks if the function successfully adds item 2 to an empty cart. The expected result is 0, indicating success.
2	blackbox	cart with 9 items	1	0	the test case checks if the function successfully adds item 1 to a cart with 9 items (cart almost full). The expected result is 0, indicating success.
3	Blackbox	cart that is full	3	-1	the test case checks if the function correctly handles the scenario when attempting to add an item to a full cart (10 items). The expected result is -1, indicating that the cart is full and shows an error occured.
4	whitebox	cart with 5 items	11	-2	the test case checks if the function correctly handles the scenario when attempting to add item 11 to a cart (item out of range). The expected result is -2, indicating an error.
5	whitebox	cart with 5 items	-1	-1	the test case examines the behavior when item is negative. The function should return a non-zero error code since item indices should be non-negative.
6	whitebox	NULL	1	-1	the test case checks if the function correctly handles the scenario when cart is NULL. The xpected result is that it should return a non-zero error code

FUNCTION: clear				
Test #	Test Type	PARAMETERS	Expected Results	Description
		Command line input		
1	Blackbox	"vrunda\n"	input buffer cleared	the test case simulates the input buffer with characters until a newline character is encountered. The expected result is that the input buffer is cleared.
2	blackbox	"\n"	input buffer cleared	the test simulates the input buffer with a newline character immediately. The expected result is that the input buffer is cleared.
3	whitebox	"\nvrunda"	input buffer is cleared after the newline character.	thw test case simulates the input buffer with a newline character followed by additional characters. The expected result is that the input buffer is cleared after the newline character.
4	whitebox	""	no action taken, the buffer remains empty	This test case checks if the function handles the scenario when the input buffer is already empty
5	whitebox	"\n\nvrunda"	the input buffer is cleared after the first newline character, and "vrunda" is left in the buffer.	the test case checks if the function can handle the case when the input buffer contains multiple consecutive newline characters. The expected result is that it should clear the input buffer after the first newline character.

Did you find more test cases via black box or white box techniques? Do you believe adequate testing could be done with just one of the techniques? Was it easier to develop black box or white box tests? Why is one faster than the other?

to me perdonally, I found both techniques good to find more test cases bcause both of it has its own advantage. For example when using black box testing , it more likely focuses on the external behaviour of the code and it helps to confirm whether the code meets its functional requirements or not. black box helped me find typical use cases and edge cases. while white box testing focuses on the actual code and its structure. it is important when testing its boundary conditions, limits, diffrent scenarios because we cannot findout about this things in black box testing. i used white box testing to create additional test cases that consider specific conditions within the functions, such as handling of NULL pointers or empty arrays. i beleive that black box and white box testing are both essential, but white box testing can be very useful for identifying edge cases and complex logic problems.

Consider how you would set up integration tests for the functions above. What would be your general approach to creating integration tests? Do you need to create additional code to set up and run the test? How will you write code to compare the results to ensure they are correct? How much additional time do you think writing the additional code will take?

Integration tests are valuable for ensuring that the interactions between functions work as expected in a real-world context, and they contribute to overall software reliability. to set up integration tests for the functions above, I would define specific scenarios involving multiple functions, such as testing the shopping cart process. I wouldd generate test data, create test code to execute these scenarios, and compare actual results to expected results using assertions. yes, writing integration tests may require additional code to set up and run the test. The time needed for writing integration tests depends on its complexity but might take hours to few days.

Create one integration test for the combination of two functions above. Show the code you created to set up the test, execute the test and compare the result to the expected result. You only need to demonstrate one set of test data, but it should be obvious how to add more tests easily. Comment your code to point out the set up, execution, and comparison parts of the code.

```
int main(void){
*
* struct Cart cart = { {0}, 0 };
* int numProducts=6;
*
* //defining the test data
* char descriptions[][MAX_STRING_LEN + 1] =
{
"flour",
"sugar",
"bananas",
"potatoes",
"milk",
"tea"
}

//setup of the test
char productName[]= "flour";
int expectedResult=2;

//execution
int n=findString(productName, descriptions, numProducts);
int rc = add2Cart(&cart, n);

//comparison
if(n==expectedResult && rc==0){
printf("test passed");
}else {
printf("test failed");
}
return 0;
```

so here , i first did the test set up where i defined the test data such as list of the items, shopping cart and the product that we are going to search for. then secondly, i executed the findsting frunction to find the position of the item in the description array and call the add2cart function to add in the cart. thirdly, comparison, i compared the results to the actual results using the conditional statements and if thye match then the integration test passes or else it fails.