SFT 221

Workshop 2

Name:	Gyeongrok oh
ID:	119140226
Email:	goh3@myseneca.ca
Section:	NBB

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	
		str
1	Blackbox	str = "bananas"
2	Blackbox	str = "apple"
3	Blackbox	str = "watermelon"
4	Blackbox	str = "sugar"
5	Blackbox	str = ""
6	Blackbox	str = " "

7	Whitebox	str = "mango"
8	Whitebox	str = "banana"
9	Whitebox	str = "cherry"
10	Whitebox	str = "apple"

PARAMETERS		Expected Results
list	nstrings	
list = ["flour", "sugar", "bananas", "potatoes", "milk", "tea"		
	nstrings =	Return value: 2
list = ["orange", "grape", "pear"]	nstring =	Return value: -1
list = ["watermelon"]		
	nstrings =	Return value: 0
list = []	nstrings =	Return value: -1
list = ["apple", "mango", "cherry"]		
	nstrings =	Return value: -1
list = [" ", "mango", "cherry"]	nstrings =	Return value: 0

list = ["apple", "banana", "cherry"]

nstrings = Return value: -1

list = ["apple", "banana", "cherry"] nstrings = Return value: -1
list = ["apple", "banana", "cherry", "cherry", "cherry"]

nstrings = Return value: 2

list = [] nstrings = (Return value: -1

Description
This test case verifies the behavior of the findString function when the target string "bananas" is present in the list array. The str parameter is set to "bananas", the list parameter is an array of strings, and the nstrings parameter is set to 6. The expected result is that the function should return the index value 2, indicating that "bananas" is found at index 2 in the list array.
This test case examines the behavior of the findString function when the target string "apple" is not present in the list array. The str parameter is set to "apple", the list parameter contains three different strings, and the nstrings parameter is set to 3. The expected result is that the function should return -1 to indicate that the target string is not found in the list array.
This test case examines the behavior of the findString function when the target string "watermelon" is present as the only element in the list array. The str parameter is set to "watermelon", the list parameter contains only the string "watermelon", and the nstrings parameter is set to 1. The expected result is that the function should return the index value 0, indicating that "watermelon" is found at index 0 in the list array, if the all parameters is the equal.
This test case checks the behavior of the `findString` function when the `list` array is empty. The `str` param
This test case examines the behavior of the findString function when the target string is an empty string. The str parameter is set to an empty string, the list parameter contains three different strings, and the nstrings parameter is set to 3. The expected result is that the function should return -1, indicating that the target string (empty string) is not found in the list array.
This test case examines the behavior of the findString function when the target string is a single whitespace character and is present as the first element in the list array. The str parameter is set to " ", the list parameter contains three strings including a whitespace character, and the nstrings parameter is set to 3. The expected result is that the function should return the index value 0, indicating that the target string (whitespace) is found at index 0 in the list array.

The loop condition i < nstrings && !found is evaluated. The loop condition evaluates to true because both i is less than nstrings and found is false. The loop body is executed. It involves checking if the current element in the list matches the target string str. After the loop body, the loop condition i < nstrings && !found is evaluated again. However, this time, the condition i < nstrings is false because i has reached a value equal to or greater than nstrings. Therefore, the loop is not executed again. This test case covers a scenario where the target string "mango" is not found in the list array. The expected result is that the function should return -1 to indicate the absence of the target string.

The loop condition i < nstrings && !found is evaluated. The loop condition evaluates to false because either i is not less than nstrings or found is true. Therefore, the loop is not executed. The function skips over the loop body. so Loop executed once. This test case covers a scenario where the nstrings parameter is set to 1, resulting in the loop being executed only once. The target string "banana" is found in the list array, but the loop terminates before reaching it. The expected result is that the function should return -1.

The loop condition i < nstrings && !found is evaluated. The loop condition evaluates to false because either i is not less than nstrings or found is true. Therefore, the loop is not executed. The function skips over the loop body. so Loop executed multiple times. This test case covers a scenario where the nstrings parameter is set to 5, and the target string "cherry" is present multiple times in the list array. The loop iterates through all elements and finds the target string at index 2. The expected result is that the function should return the index value 2.

The loop condition i < nstrings && !found is evaluated. The loop condition evaluates to false because either i is not less than nstrings or found is true. Therefore, the loop is not executed. The function skips over the loop body. so Loop executed zero times. This test case covers a scenario where the nstrings parameter is set to 0, resulting in an empty list array. Since there are no elements in the array, the loop is not executed. The expected result is that the function should return -1 to indicate the absence of the target string.

eter is set to "sugar", the `list` parameter is an empty array, and the `nstrings` parameter is set to 0. The expect

:ed result is that the function should return -1, indicating that the target string is not found in the empty array

FUNCTION: init

Test #	Test Type	ar
1	Blackbox	[0, 0, 0]
2	Blackbox	[!,@,#,$]
3	Blackbox	[]
4	Blackbox	[-1, -2, -3, -4, -5]
5	Blackbox	[1, 2, 3, 4, 5],
6	Blackbox	[0, 0, 0, 0]
7	Whitebox	[0, 0, 0, 0, 0, ... (repeated MAX_CART ti
8	Whitebox	[0, 0, 0]
9	Whitebox	[0,0,0]
10	Whitebox	ar = NULL

PARAMETERS		Expected Results
value	size	
value = -1		
	size = 3	[-1, -1, -1]
value = 0	size = 4	[0,0,0,0]
value = 5		
	size = 0	[]
value = -1	size = 5	[-1, -1, -1, -1, -1]
value = 0		
	size = 0	[1, 2, 3, 4, 5]
value = 1	size = 3	[1, 1, 1, 0]
value = 1,		
	size = MA	[1, 1, 1, 1, 1, ... (r
value = 2		
	size = -5	Error
value = !		
	size - 3	Error
value = 3		
	size = 4	Error

Description
Testing the init function by providing an array ar of size 3 that is initialized with zeros. The value parameter is set to -1. The expected result is that all elements in the array should be set to -1.
Testing the init function by providing an array ar of size 4 that is initialized with special characters. The value parameter is set to 0. The expected result is that all elements in the array should be set to 0, even if inside of the ar have a special characters.
Testing the init function with an empty array ar and size 0. The expected result is that the array should remain empty.
This test case verifies the behavior of the init function when the array ar is initialized with negative numbers. The value parameter is set to -1, and the size parameter is 5. The expected result is that all elements in the array should be set to -1.
Testing the init function with an array ar of size 5 initialized with consecutive numbers. The size parameter is set to 0. The expected result is that the array should remain unchanged.
This test case verifies the behavior of the init function when the array ar is initialized with zeros. The value parameter is set to 1, and the size parameter is 3. The expected result is that the first three elements in the array should be set to 1, while the last element should remain unchanged.
This test case aims to test the performance of the init function with a large array size (MAX_CART). The init function should initialize all elements in the array to the value 1 without any performance or memory issues.
This test case covers the scenario when the size parameter is a negative value. The init function should handle this error condition gracefully, either by returning an error code or throwing an exception.
When we call the init function, we pass the value as the second argument. However, the value parameter in the init function has an int data type. If we attempt to pass a character, such as '!', as the value argument, it would result in a compilation error in the code. The data types of the arguments need to match the expected data types of the function parameters for the code to compile without errors.
This test case verifies the error handling capability of the init function when the array pointer ar is NULL. The value parameter is set to 3, and the size parameter is 4. The expected result is that the function should handle the NULL pointer gracefully, either by returning an error code or throwing an exception.

FUNCTION: add2Cart

Test #	Test Type	cart
1	Blackbox	cart: { items: [0, 0, 0, 0], nItems: 0 }
2	Blackbox	{ items: [1, 3, 5, 7, 9], nItems: 5 }
3	Blackbox	cart: { items: [2, 4, 6, 8], nItems: 4
4	Blackbox	cart: { items: [0, 0, 0], nItems: 3 }
5	Blackbox	cart: { items: [0, 1, 2, 3, 4], nItems: 5 }
6	Blackbox	cart: { items: [1, 3, 5, 7, 9], nItems: 5 }
7	Whitebox	cart: { items: [0, 0, 0], nItems: 0 }
8	Whitebox	cart: { items: [0, 0, 0, 0, 0], nItems: 10 }
9	Whitebox	cart: { items: [0, 0, 0], nItems: 0 }
10	Whitebox	cart: { items: [1, 2, 3, 4, 5, 6, 7, 8, 9, 0], nItems: 10 }

PARAMETERS	Expected Results
item	
item: 2	
	cart: { items: [0, 0, 2,
item: 4	cart: { items: [1, 3, 5,
item: -1	
	cart: { items: [2, 4, 6,
item: 10	cart: { items: [0, 0, 0]
item: 5	
	cart: { items: [0, 1, 2,
item: 10	cart: { items: [1, 3, 5,
item: -1	
	cart: { items: [0, 0, 0]
item: 3	cart: { items: [0, 0, 0,
item: 0	
	cart: { items: [0, 0, 0,
item: 5	cart: { items: [1, 2, 3,

Description
Testing the normal flow of adding an item to the cart. The initial cart is empty with nItems set to 0. The item to be added is 2. The expected result is that the item is added to the cart, and nItems is incremented to 1.
Testing the boundary case where the cart already has the maximum number of items (MAX_CART) before adding a new item. The initial cart has items [1, 3, 5, 7, 9] and nItems set to 5. The item to be added is 4. The expected result is that the item is added to the cart, and nItems is incremented to 6.
Testing the case where an invalid item is provided (item < 0). The initial cart has items [2, 4, 6, 8] and nItems set to 4. The item to be added is -1. The expected result is that the cart remains unchanged.
Testing the case where an item that exceeds the maximum index of the items array (MAX_PRODUCTS) is provided. The initial cart has items [0, 0, 0] and nItems set to 3. The item to be added is 10. The expected result is that the cart remains unchanged.
Testing the boundary case where the cart has one less than the maximum capacity (MAX_CART - 1) before adding a new item. The initial cart has items [0, 1, 2, 3, 4] and nItems set to 5. The item to be added is 5. The expected result is that the item is added to the cart, and nItems is incremented to 6.
Testing the case where an item that exceeds the maximum index of the items array (MAX_PRODUCTS) is provided. The initial cart has items [1, 3, 5, 7, 9] and nItems set to 5. The item to be added is 10. The expected result is that the cart remains unchanged.
This test case verifies the behavior when an invalid item value is provided (less than 0). The function should handle the exception and not add the item to the cart. It ensures that the condition if (item >= 0 && item < MAX_PRODUCTS) evaluates as false.
This test case covers the scenario where the cart has exceeded its maximum capacity (MAX_CART). The function should handle the exception and not add the item to the cart. It ensures that the condition if (cart->nItems >= MAX_CART) evaluates as true.
This test case verifies the behavior when adding the first item to an empty cart. It ensures that the statement cart->items[cart->nItems] = item is executed, and the item is added to the cart.
This test case ensures that the function handles the scenario when the cart is already full. The item should not be added to the cart, and the cart remains unchanged.

FUNCTION: clear		
Test #	Test Type	PARAMETERS
		Command line input
1	Blackbox	Enter item to purchase (end to stop): "apple "
2	Blackbox	Enter item to purchase (end to stop): "apple "
3	Blackbox	Enter item to purchase (end to stop): "apple123!@#"
4	Blackbox	Enter item to purchase (end to stop): ""
5	Blackbox	Enter item to purchase (end to stop): "apple\"
6	Blackbox	Enter item to purchase (end to stop): "12345"
7	Whitebox	Enter item to purchase (end to stop): "A "
8	Whitebox	Enter item to purchase (end to stop): "apple banana "
9	Whitebox	Enter item to purchase (end to stop): "\n"
10	Whitebox	Enter item to purchase (end to stop): "apple\banana\melon "

Expected Results	Description
Input buffer cleared	This test case ensures that the clear function correctly clears the input buffer, excluding any leading whitespace characters before the actual input.
Input buffer cleared	This test case verifies that the clear function properly clears the input buffer, excluding any trailing whitespace characters after the actual input.
Input buffer cleared	This test case ensures that the clear function handles and clears the input buffer correctly, even when the input contains special characters and numbers.
No action taken, as t	This test case verifies that the clear function handles empty input correctly. Since there is no input to clear, the function should not perform any actions and simply return.
Input buffer cleared	This test case ensures that the clear function handles input lines that end with a backslash correctly. The function should clear the input buffer until the last character, including the backslash itself.
Input buffer cleared	This test case ensures that the clear function handles input lines with numeric values correctly. The function should clear the input buffer until the newline character, disregarding any numeric values present in the input.
The input buffer is cl	This test case verifies that the clear function correctly handles an empty input buffer by not performing any action.
The input buffer is cl	This test case checks the behavior of the clear function when there are multiple characters in the input buffer. It ensures that the function removes all characters from the buffer until it reaches the newline character.
The newline charact	This test case validates that the clear function handles the presence of a newline character in the input buffer correctly. The function should remove the newline character from the buffer.
The input buffer is c	This test case verifies that the clear function correctly clears the input buffer when it contains multiple lines. It should remove all characters and newline characters from the buffer until the newline character is encountered.

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?

To set up an integrated test for a function, you can design a test case in which multiple functions interact. This needs to make a test scenario in which the output of one function is used as the input of another function. The expected output is predetermined to compare results and ensure accuracy, and the actual output is compared to the desired output using an approximation or comparison technique. Additional code may be required to set up a test domain, develop test data, and initialize variables or structures. The time needed to write different codes for integrated testing may vary depending on the complexity of the feature and the number of test cases.

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?

I found more cases via Blackbox. For effective testing, both black and white box techniques are recommended. Each course provides unique insights. Black-box testing helps ensure the system meets requirements and works correctly from a user's perspective. In contrast, white-box testing helps identify problems by validating internal logic and code implementations. Blackbox testing is easier to develop and faster to run because it focuses on external behavior and does not inspect code. White-box testing requires more effort and technical understanding to analyze internal code and design tests for specific paths or situations. However, the white box test allows you to control and verify internal behavior in detail, resulting in more accurate results.

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.

```
#include <stdio.h>
#include <string.h>

#define MAX_STRING_LEN 25
#define MAX_PRODUCTS 10
#define MAX_CART 10

struct Cart
{
   int items[MAX_CART];
   int nItems;
};

int add2Cart(struct Cart* cart, const int item)
{
   // Implementation of add2Cart function

}
void clear()
```

In the "Setup" part of the code, necessary variables and test data are declared and initialized, including the ca

In the "Execution" part, the code calls the findString function to find the index of productName in the descrip

In the "Comparison" part, assertions are used to compare the actual results to the expected results. It verifies rt structure, prices, descriptions, numProducts, and productName.

tions array. Then, it calls the add2Cart function to add the item to the cart, and finally, the clear function is invo

if the return value of add2Cart is 0, indicating that the item was successfully added to the cart, and checks if th

›ked to clear the input buffer.

ıe nItems member of the cart structure is 1, confirming that the cart contains one item.