Circular Linked List Test Plan

Develop the test plan: *For each member function that you plan to write, think about how to test it – what flow of control exists in the member function and how would you test out all conditions:*

Test Case(s)	Expected Result	Verified? (yes/no)
Push_back:		
On an empty list	Element is added, rear and front point to the same element	**Yes**
On a populated list	Element is added to the rear of the list, rear pointer is updated	**Yes**
Pop_front:		
On an empty list with no out parameter	Nothing, list is unchanged	**Yes**
On an empty list with an out parameter	Parameter is returned unchanged	**Yes**
On a list with one element	Element is deleted, parameter overwritten if provided, list is empty	**Yes**
On a list with multiple elements	Front is deleted, parameter overwritten if provided, front point is updated	**Yes**
Clear:		
On an empty list	Nothing, list is unchanged	**Yes**
On a populated list	Each node is destroyed, pointers are set to NULL	**Yes**
Size:		
On an empty list	Returns 0	**Yes**
On a list with 'N' elements	Returns N	**Yes**
Empty:		
On an empty list	Returns true	**Yes**
On a populated list	Returns false	**Yes**

Verify correctness: Using the above test plan, create a test program that tests the interactions of all functions together.

Linear Linked List Test Plan

Develop the test plan: *For each member function that you plan to write, think about how to test it – what flow of control exists in the member function and how would you test out all conditions:*

Test Case(s)	Expected Result	Verified? (yes/no)
Push_front:		
On an empty list	Element is added to the front, head and tail point to the same element	**Yes**
On a populated list	Element is added to the front of the list, head pointer is updated	**Yes**
Pop_front:		
On an empty list with no out parameter	Nothing, list is unchanged	**Yes**
On an empty list with an out parameter	Parameter is returned unchanged	**Yes**
On a list with one element	Element is deleted, parameter overwritten if provided, list is empty	**Yes**
On a list with multiple elements	Front is deleted, parameter overwritten if provided, head pointer is updated	**Yes**
Clear:		
On an empty list	Nothing, list is unchanged	**Yes**
On a populated list	Each node is destroyed, pointers are set to NULL	**Yes**
Size:		
On an empty list	Returns 0	**Yes**
On a list with 'N' elements	Returns N	**Yes**
Empty:		
On an empty list	Returns true	**Yes**
On a populated list	Returns false	**Yes**

Verify correctness: Using the above test plan, create a test program that tests the interactions of all functions together.

Feature_Queue Test Plan

Develop the test plan: *For each member function that you plan to write, think about how to test it – what flow of control exists in the member function and how would you test out all conditions:*

Test Case(s)	Expected Result	Verified? (yes/no)
enqueue:		
On an empty queue	Feature string is added to the queue, front and rear point to the new element	**Yes**
On a populated queue	Element is added to the rear of the queue, rear pointer is updated	**Yes**
dequeue:		
On an empty queue	Returns false, out parameter is unchanged	**Yes**
On a queue with one element	Returns true, out parameter is overwritten, queue is now empty	**Yes**
On a populated queue	Returns true, out parameter is overwritten	**Yes**
Peek_front/Peek_back:		
On an empty queue	Nothing, queue is empty	**Yes**
On a populated queue	A read only reference is returned to the front/rear of the queue	**Yes**
Clear:		
On an empty queue	Nothing, list is unchanged	**Yes**
On a populated queue	Each item is dequeued and set to NULL	**Yes**
Size:		
On an empty queue	Returns 0	**Yes**
On a queue with 'N' elements	Returns N	**Yes**
Empty:		

On an empty queue	Returns true	**Yes**
On a populated queue	Returns false	**Yes**
Display_all:		
On an empty queue	Nothing is displayed	**Yes**
On a populated queue	Each item is displayed in a formatted fashion	**Yes**

Verify correctness: Using the above test plan, create a test program that tests the interactions of all functions together.

Product_Stack Test Plan

Develop the test plan: *For each member function that you plan to write, think about how to test it – what flow of control exists in the member function and how would you test out all conditions:*

Test Case(s)	Expected Result	Verified? (yes/no)
push:		
On an empty stack	Product is pushed to the top of the stack, a single node with 5 elements is instantiated	**Yes**
Pushing a 6th Product	Product is pushed to the top of the stack, a new node with another 5 elements is pushed to the front of the list, top_index is reset	**Yes**
pop:		
On an empty stack	Returns false, out parameter is unchanged	**Yes**
On a stack with one element	Returns true, out parameter is overwritten, stack is now empty	**Yes**
On a stack with 6 elements	Returns true, out parameter is overwritten, the empty node is deleted, top_index is rolled back up to 5	**Yes**
Peek_top/Peek_bottom:		
On an empty stack	Nothing, queue is empty	**Yes**
On a populated queue	A read only reference is returned to the top/bottom of the queue	**Yes**
Clear:		
On an empty stack	Nothing, list is unchanged	**Yes**

On a populated stack	Each array on each node is deleted along with each node	**Yes**
Size:		
On an empty stack	Returns 0	**Yes**
On a stack with 'N' elements	Returns N	**Yes**
Empty:		
On an empty stack	Returns true	**Yes**
On a populated stack	Returns false	**Yes**
Display_all:		
On an empty stack	Nothing is displayed	**Yes**
On a populated stack	Each item is displayed in a formatted fashion	**Yes**

Verify correctness: Using the above test plan, create a test program that tests the interactions of all functions together.