Assignment Three - Mahnoor Shahid

Screenshots - Question One

1. I began by creating a queue for integers and adding 10 values to it. Next, display the elements in the queue using only the queue functions and this was achieved with the push, pop, front, and size functions in the code.

```
The current elements in queue: 1 2 3 4 5 6 7 8 9 10
```

2. I created a move_to_rear function that takes the first element in the queue and moves it to the back. The element that was second in line becomes the new front element. For example, 1 is at the front, after using this function, 1 goes to the rear and 2 moves to the front. I called the function again, 2 goes to the rear and 3 is now at the front. This process keeps going with each function call.

```
The queue after moving the front element to rear (first time): 2 3 4 5 6 7 8 9 10 1
The queue after moving the front element to rear (second time): 3 4 5 6 7 8 9 10 1 2
The queue after moving the front element to rear (third time): 4 5 6 7 8 9 10 1 2 3
```

Screenshots - Question Two

1. I changed the linear search function to find the last occurrence of a target in a vector instead of the first one. I have a vector that looks like this:
   vector<int> items = {1, 2, 3, 4, 2, 5, 6, 2};
   I applied the function with my target being 2, and it correctly identifies the last occurrence at index 7.

```
The last occurrence of 2 is at index: 7
```

   Now, I will look for a target that isn't in the vector. For example, since 8 is not in the vector, it should display an error message indicating that the target was not found.

```
Error: Target not found in the vector.
```

Screenshots - Question Three

1. I changed the insertion sort function so that it can sort a list of integers using a singly linked list. I have a linked list that looks like this:
   Node* head = new Node(5);
   head->next = new Node(4);
   head->next->next = new Node(3);
   head->next->next->next = new Node(2);
   head->next->next->next->next = new Node(1);
   When we print this list, it appears unsorted and should display from 5 down to 1.

   ```
   Original List (before sorting): 5 4 3 2 1
   ```

2. I will now call the insertion sort function that was created with a linked list to sort this list in ascending order. It should print the numbers from 1 to 5 sorted in the right order.

   ```
   Sorted List (after sorting): 1 2 3 4 5
   ```

How to run the code:

Question One - This code demonstrates the use of a queue with integers. It first instantiates a queue and pushes 10 values (1 through 10) into it. It then displays the current elements in the queue. The program proceeds to move the front element to the rear of the queue three times, displaying the updated queue after each move to show how the elements shift. This highlights how the ‘move_to_rear’ function works by cycling the front element to the back of the queue.

Question Two - This code demonstrates the use of a recursive function to find the last occurrence of a target value in a vector. It starts from the last index and searches backwards, returning the index of the last occurrence of the target. If the target isn’t found, it returns -1.

Question Three - This code demonstrates the insertion sort algorithm on a singly linked list. The ‘insertion_sort’ function iterates through the original list, inserting each node into its correct position in a new sorted list. It checks if the node should be placed at the beginning or finds the appropriate position in the sorted list. After processing all nodes, it updates the ‘head’ pointer to the sorted list. The ‘print_list’ function is used to display the list before and after sorting.