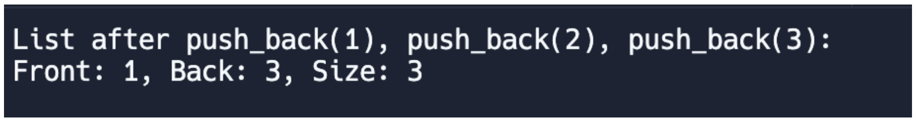
Assignment Two - Mahnoor Shahid

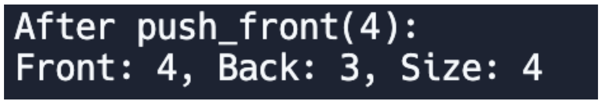
Screenshots - Question One

1. You can add items to the end of the list. For instance, I used push_back to add the numbers 1, 2, and 3 to the list. After that, it shows the number at the front of the list, the number at the back, and the total size of the list.



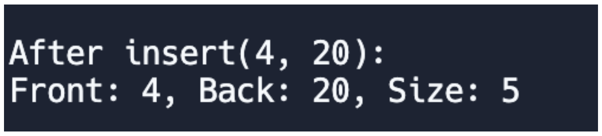
```
List after push_back(1), push_back(2), push_back(3):
Front: 1, Back: 3, Size: 3
```

2. You can add items to the start of the list. For example, with the numbers 1, 2, and 3, I add the number 4 at the front. Now, 4 is at the beginning of the list instead of 1. The size of the list also grows because a new item was added.



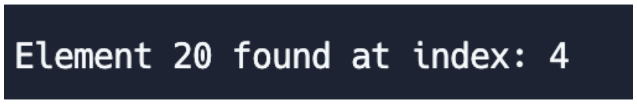
```
After push_front(4):
Front: 4, Back: 3, Size: 4
```

3. You can add items at a specific index. For instance, I am adding 20 at index 4, which is the end of the list right now. Now, 20 is at index 4, and the list size grows because we added a new item.



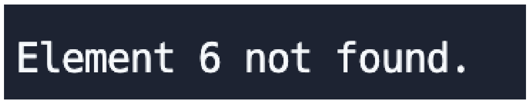
```
After insert(4, 20):
Front: 4, Back: 20, Size: 5
```

4. You can locate an item in the list if it is present. I am looking for the number 20, and since it is in the list, it tells me the index where that number is found.



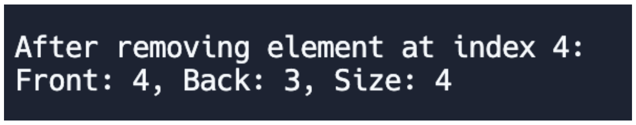
```
Element 20 found at index: 4
```

If you look for an item that isn't in the list, it will indicate that it wasn't found. For instance, the number 6 is not in the list, so it won't be found.



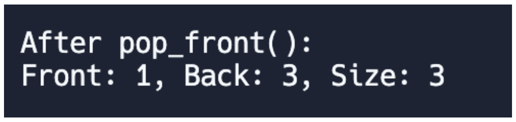
```
Element 6 not found.
```

5. You can remove an item from a specific index. For instance, I'm removing the item at index 4, which is 20. When it's removed, the list will become smaller.



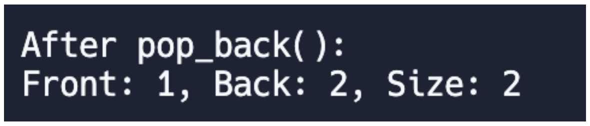
```
After removing element at index 4:
Front: 4, Back: 3, Size: 4
```

6. You can remove items from both the front and back of the list. First, I’m removing the item at the front, which was 4, so now 1 is at the front. The size of the list gets smaller.



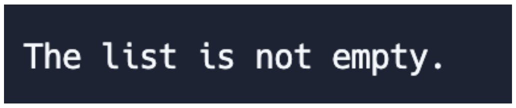
```
After pop_front():
Front: 1, Back: 3, Size: 3
```

   Next, I’m removing the item from the back, which is 3. When I remove it, 2 is now at the back and the size of the list will also get smaller.



```
After pop_back():
Front: 1, Back: 2, Size: 2
```

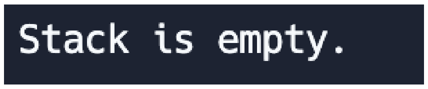
7. You can check if the list has any items - empty or not. Currently, the list has 2 elements, 1 and 2, so it will show that it is not empty.



```
The list is not empty.
```

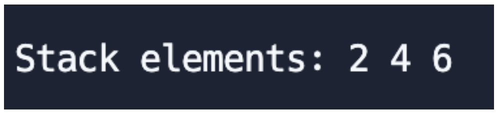
Screenshots - Question Two

1. You can see if the stack is empty. Right after I made my stack object, I checked if it was empty. It is empty since no numbers have been added to it yet.



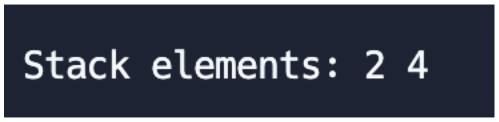
```
Stack is empty.
```

2. You can add integers to the stack. Here, I am inserting the numbers 2, 4, and 6, so the stack shows the values that are currently present.



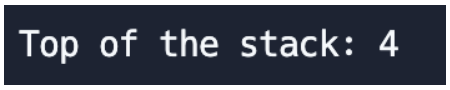
```
Stack elements: 2 4 6
```

3. You can remove an element from the stack. Here, I am removing the value 6 and the only values left are 2 and 4.



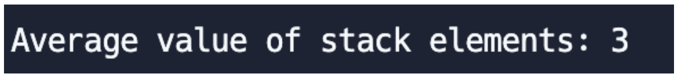
```
Stack elements: 2 4
```

4. You can see the top of the stack. I removed 6 in the last example, which was at the top, so now 4 is the new top.



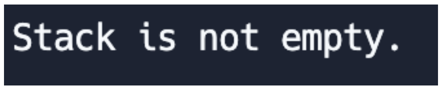
```
Top of the stack: 4
```

5. You can calculate the average of the numbers in the stack. In my stack, there are just two numbers: 4 and 2. If you add them together, 4 + 2 equals 6. Then, divide 6 by 2, which gives you 3. So, the average of these two numbers is 3.



```
Average value of stack elements: 3
```

6. I am checking if the stack is empty again. There are still two elements, 4 and 6, in the stack, so it is not empty.



```
Stack is not empty.
```

How to Run the Code:
Question 1 - This C++ code demonstrates various operations on a singly linked list, including adding elements to the front and back of the list, inserting an element at a specific index, and removing elements. It also includes functionality for finding an element in the list and checking its size. Additionally, the code removes elements from both the front and back, checks if the list is empty, and prints the list's front, back, and size at several points to show the changes as different operations are performed.

Question 2 - This C++ code demonstrates various operations on a stack. It begins by checking if the stack is empty, then proceeds to push several elements onto it. After displaying the stack's contents, an element is removed, and the stack is displayed again. The code then retrieves and prints the top element of the stack and calculates the average value of the elements. Finally, it checks once more if the stack is empty.