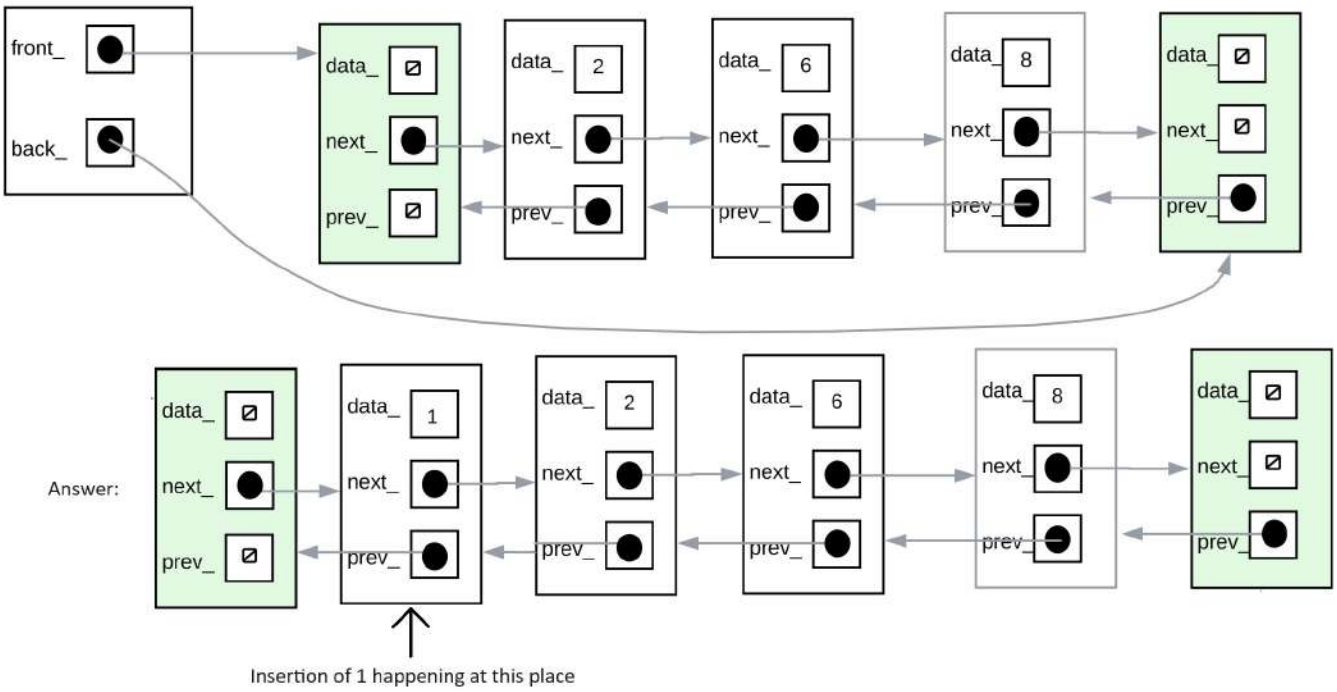
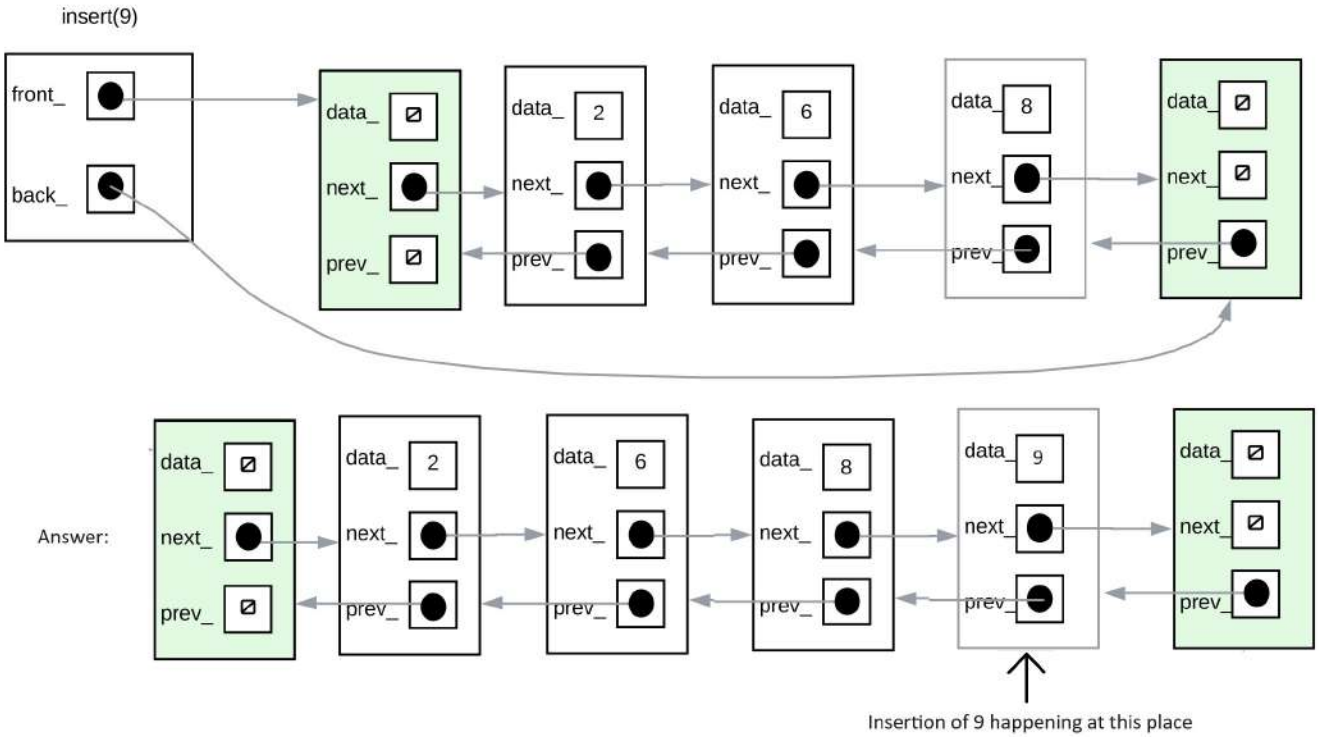
insert(1)

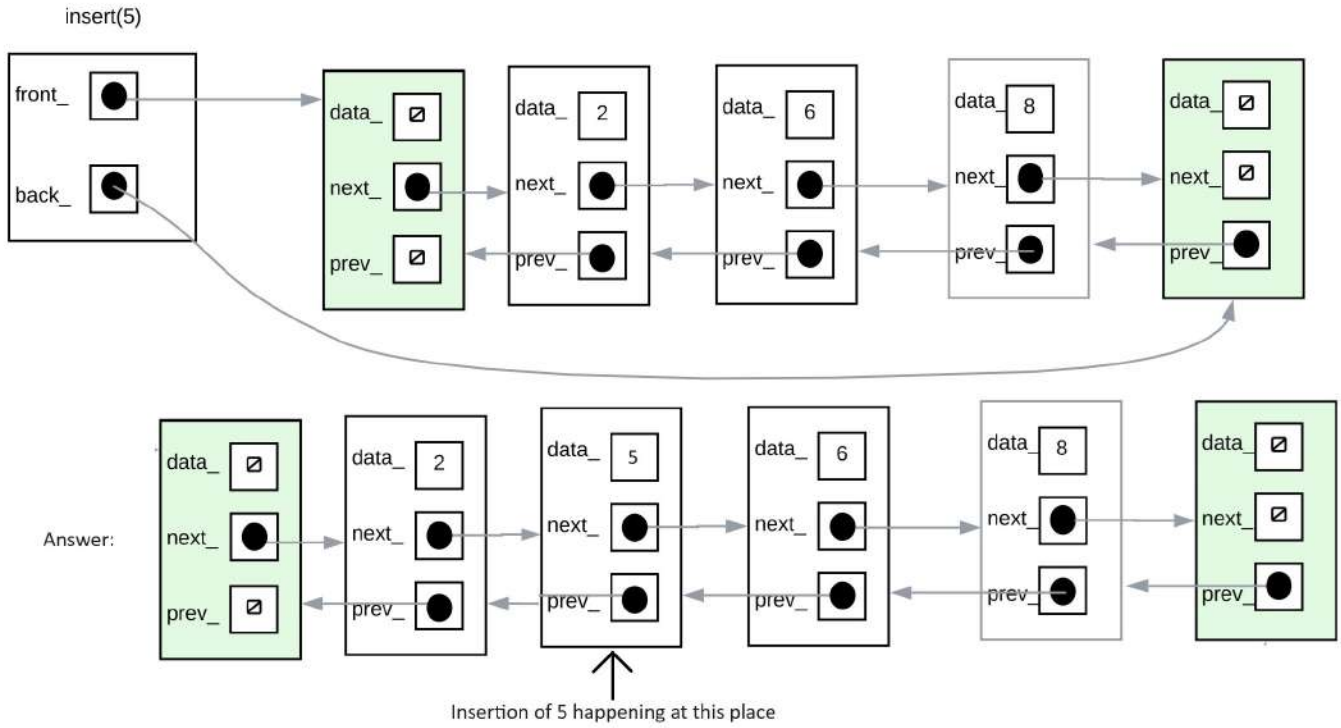



For executing insertion of 1 in the linked list, we would do the following:
1. update the next_ address of front sentinel node to the current address of 1.
2. add the prev_ address of 1 to the current address of front sentinel node and the next_ address of 1 to the current address of 2.
3. update the prev_ address of 2 to the current address of 1.

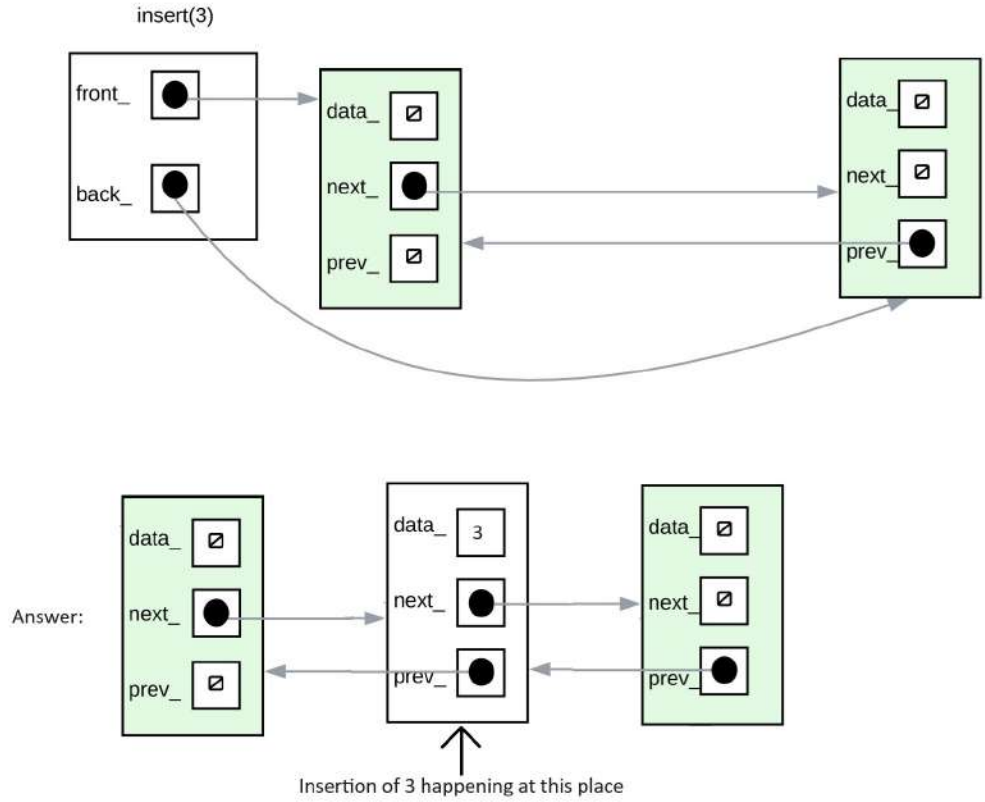
insert(9)



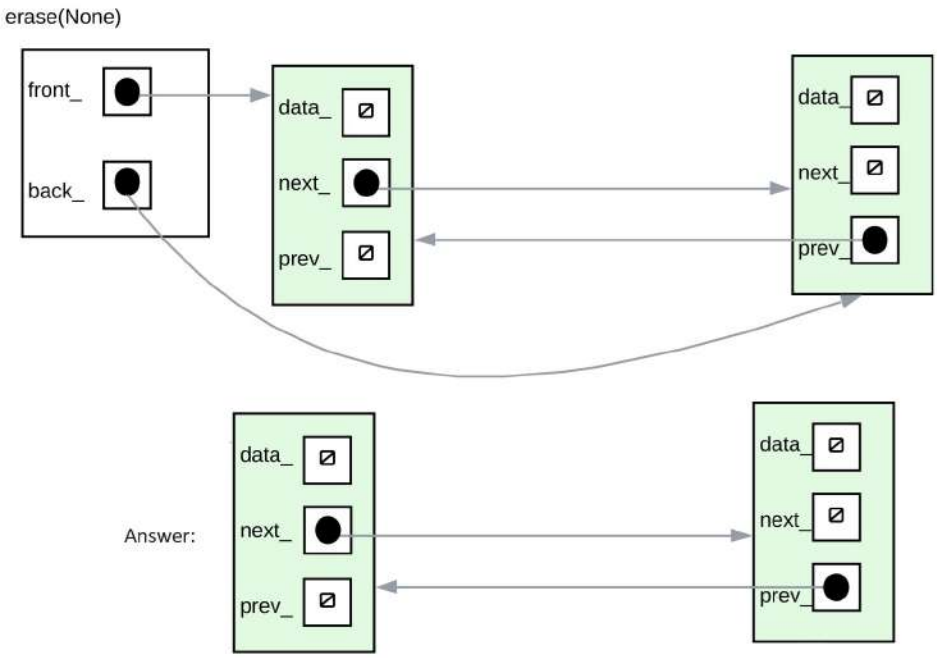

For executing insertion of 9 in the linked list, we would do the following:
1. update the next_ address of 8 to the current address of 9.
2. add the prev_ address of 9 to the current address of 8 and the next_ address of 9 to the current address of back sentinel node.
3. update the prev_ address of back sentinel node to the current address of 9.



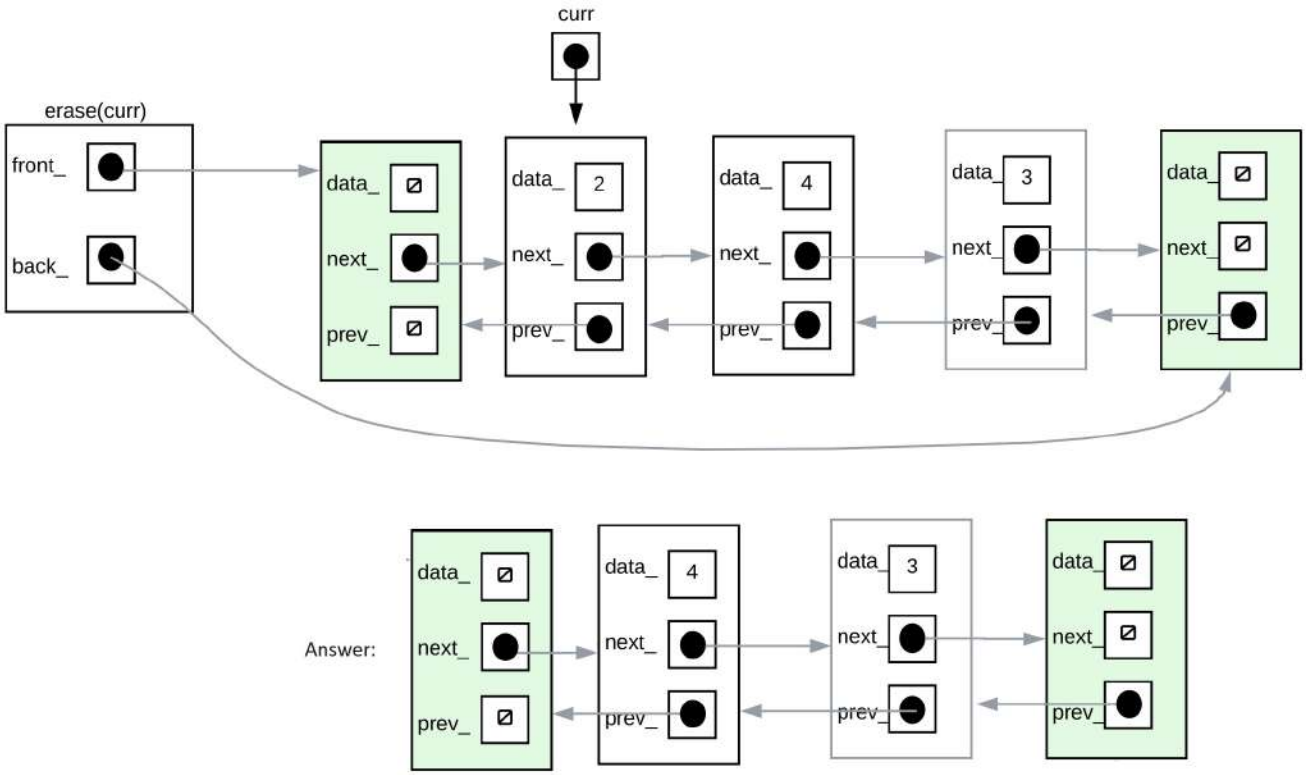

For executing insertion of 5 in the linked list, we would do the following:
1. update the next_ address of 2 to the current address of 5.
2. add the prev_ address of 5 to the current address of 2 and the next_ address of 5 to the current address of 6.
3. update the prev_ address of 6 to the current address of 5.



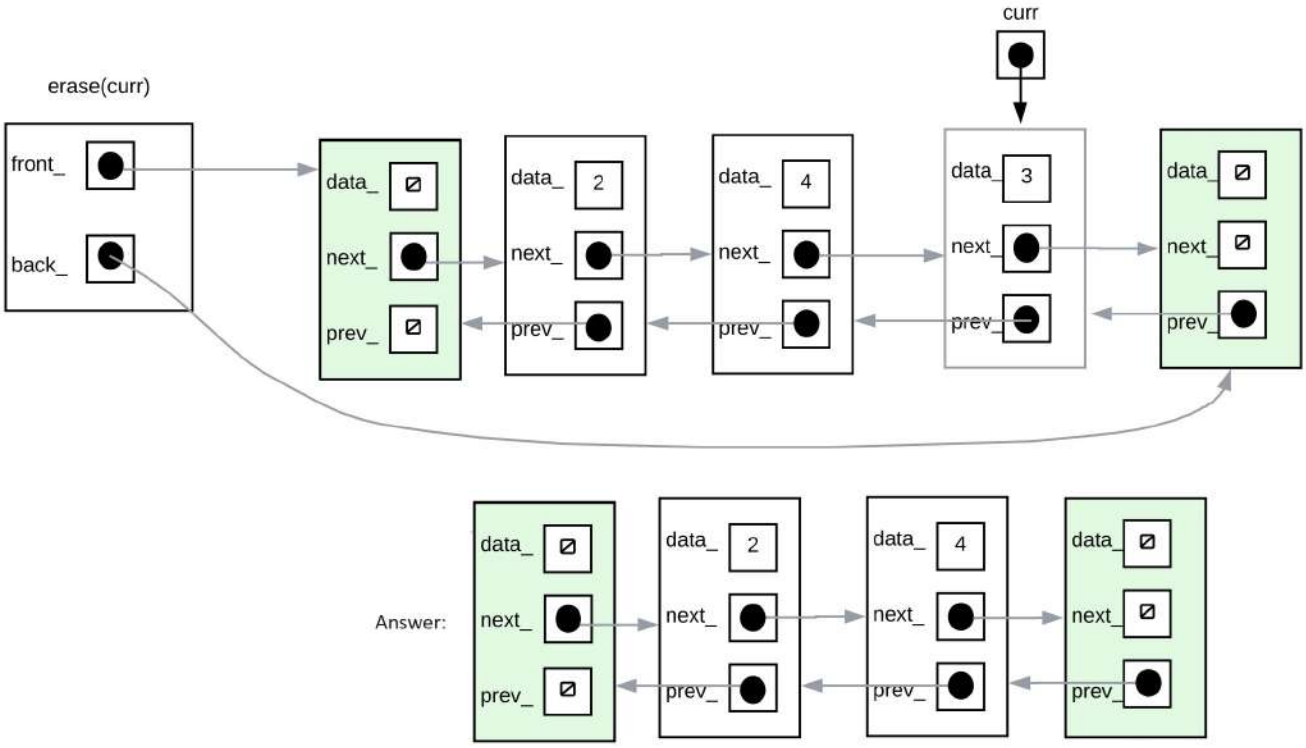

For executing insertion of 3 in the linked list, we would do the following:
1. update the next_ address of front sentinel node to the current address of 3.
2. add the prev_ address of 3 to the current address of front sentinel node and the next_ address of 3 to the current address of back sentinel node.
3. update the prev_ address of back sentinel node to the current address of 3.



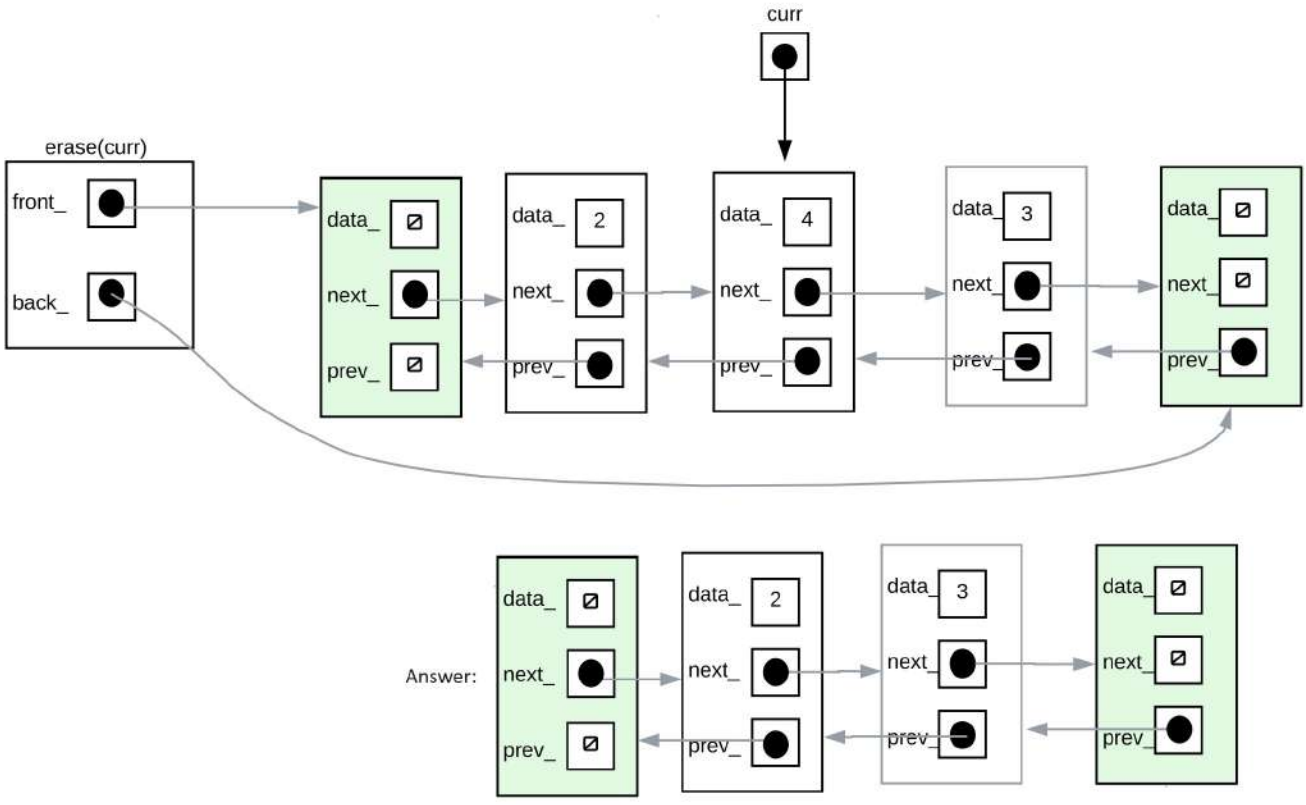

Error: Node value referred as None cannot be deleted



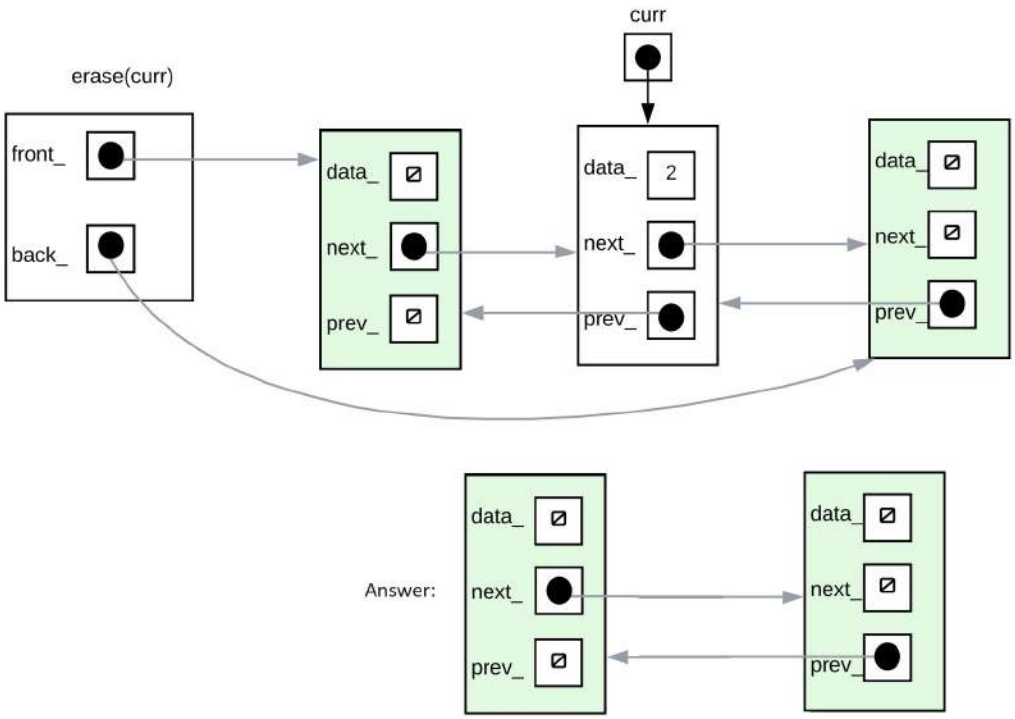

For executing deletion of 2 in the linked list, we would do the following:
1. update the next_ address of front sentinel node to the current address of 4.
2. update the prev_ address of 4 to the current address of front sentinel node.
3. delete the node containing 2.




For executing deletion of 3 in the linked list, we would do the following:
1. update the next_ address of 4 to the current address of back sentinel node.
2. update the prev_ address of back sentinel node to the current address of 4.
3. delete the node containing 3.




For executing deletion of 4 in the linked list, we would do the following:
1. update the next_ address of 2 to the current address of 3.
2. update the prev_ address of 3 to the current address of 2.
3. delete the node containing 4.




Answer:

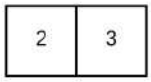



For executing deletion of 2 in the linked list, we would do the following:
1. update the next_ address of front sentinel node to the current address of back sentinel node.
2. update the prev_ address of back sentinel node to the current address of front sentinel node.
3. delete the node containing 2.

Stack: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

stack.push(6)

3 is at top of stack

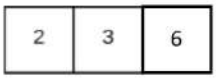



Initial State: The stack starts with the array [2, 3].

After stack.push(6): Adding 6 to the top updates the array to [2, 3, 6].

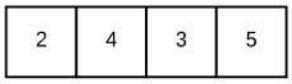
Top of Stack: 6 is at the top of the stack.

Final Stack




stack.pop()
stack.pop()
stack.push(6)

initially 5 is at top of stack

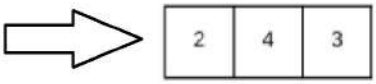



Initial State:

The stack starts with the array [2, 4, 3, 5], a top index of 3, and a capacity of 4.

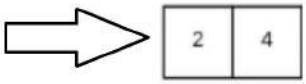
After stack.pop():

Removing the top element updates the array to [2, 4, 3, None] and the top index to 2, with capacity remaining 4.



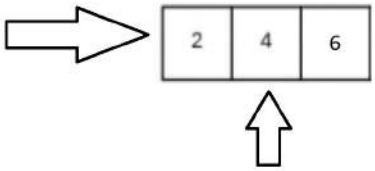

After stack.pop():

Removing the next top element updates the array to [2, 4, None, None] and the top index to 1, with capacity still 4.




After stack.push(6):

Pushing 6 updates the array to [2, 4, 6, None] and the top index to 2, with no change in capacity.

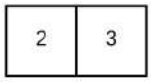



Final Stack

Queues: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

queue.enqueue(6)

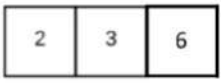
2 is at front of queue, 3 is at back




Initial State: The queue starts with elements 2 at the front and 3 at the back.

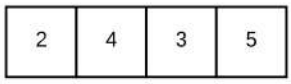
After queue.enqueue(6): Adding element 6 to the back of the queue updates it to [2, 3, 6].

Final Queue



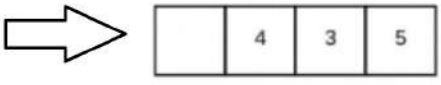

queue.dequeue()
queue.dequeue()
queue.enqueue(6)

initially 2 is at front of queue, 5 is at back

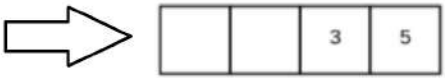



Initial State: The queue starts with the array [2, 4, 3, 5].

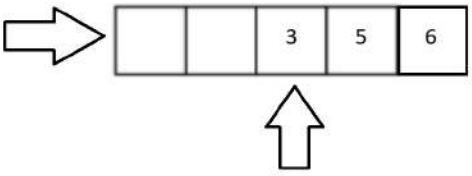
After queue.dequeue(): Removing the first element updates the array to [null, 4, 3, 5].




After another queue.dequeue(): Removing the first element updates the array to [null, null, 3, 5].




After queue.enqueue(6): Adding 6 at the end updates the array to [null, null, 3, 5, 6].

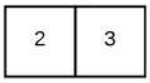



Final Queue

Deques: In the diagrams below list what data members you need to track and what their values are in its initial state and their state after each of the operations are applied to the diagram. If the array needs to be resized, draw the new array with the correct capacity

deque.push_front(6)

2 is at front of Deque, 3 is at back

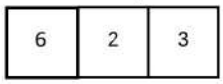



Explanation:

After deque.push_front(6):

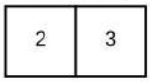
Adding 6 to the front of the deque necessitates resizing the array to accommodate the new element. The array becomes [6, 2, 3], the front index shifts to 2, and the back index remains at 1. The capacity increases to 4 to accommodate potential future additions.

Final Deque




deque.push_back(6)

2 is at front of Deque, 3 is at back

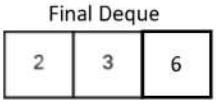



Explanation:

After deque.push_back(6):

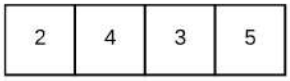
Adding 6 to the back updates the array to [2, 3, 6], the front index remains at 0, the back index becomes 2, and the capacity increases to 4.

Final Deque




deque.pop_back()
deque.push_front(6)

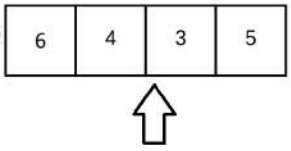
initially 2 is at front of deque, 5 is at back




Initial State: The deque starts with the array [2, 4, 3, 5].

After deque.pop_back(): Removing the last element updates the array to [2, 4, 3].

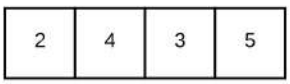

After deque.push_front(6): Adding 6 at the front updates the array to [6 2, 4, 3].




Final Deque

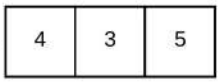
deque.pop_front()
deque.push_back(6)
deque.pop_front()
deque.push_back(7)

initially 2 is at front of deque,
5 is at back

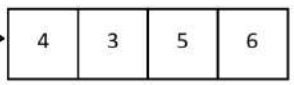



Initial State: The deque starts with the array [2, 4, 3, 5].

After deque.pop_front(): Removing the first element updates the array to [4, 3, 5].

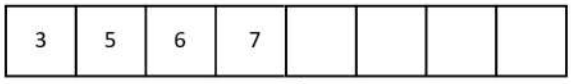



After deque.push_back(6): Adding 6 at the end updates the array to [4, 3, 5, 6].




After deque.pop_front(): Removing the first element updates the array to [3, 5, 6].


After deque.push_back(7): Adding 7 at the end updates the array to [3, 5, 6, 7]. Resizing of the array to capacity 8 was necessary.




Final Deque

overflow(grid,the_queue) - apply the overflow function to the gride below and show all the grids the function would add to the queue. Number the grid in the order they are added to the queue. Also state the return value. Note that some grids may remain empty

| | | | | |
|---|---|---|---|---|
| -2 | 1 | -3 | -3 | 0 |
| 2 | 0 | 3 | 2 | 0 |
| 0 | 0 | -3 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | -3 | -1 | -1 | -1 |
| -3 | 0 | -4 | -3 | 0 |
| 0 | 0 | -3 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| -2 | 0 | -3 | -1 | -1 |
| 0 | -3 | 0 | -4 | 0 |
| -1 | 0 | -4 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | -2 | 0 | -3 | -1 |
| -1 | -3 | -3 | 0 | -1 |
| -1 | -1 | 0 | -2 | 0 |
| 0 | 0 | -2 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |