**1. Insert at Front (insert\_at\_front method):**

**Algorithm:**

1. Create a new node (newNode) with the given data.
2. Set the next pointer of newNode to the current head.
3. Set the prev pointer of newNode to NULL.
4. If the list is not empty (head != NULL), update the prev pointer of the current head to point to newNode.
5. Update the head to newNode.

**2. Insert at End (insert\_at\_end method):**

**Algorithm:**

1. Create a new node (newNode) with the given data.
2. If the list is empty (head == NULL), set the prev pointer of newNode to NULL and update head to newNode.
3. If the list is not empty, traverse the list to the last node (temp).
4. Set the next pointer of the last node to newNode.
5. Set the prev pointer of newNode to the last node

**4. Delete at Front (delete\_at\_front method):**

**Algorithm:**

1. Check if the list is empty. If so, print an error message.
2. Set temp to the current head.
3. Update head to the next node.
4. If the new head is not NULL, update its prev pointer to NULL.
5. Delete the original head (temp).

**5. Delete at End (delete\_at\_end method):**

**Algorithm:**

1. Check if the list is empty. If so, print an error message.
2. Set temp to the current head and initialize prev to NULL.
3. Traverse the list to the last node (temp), keeping track of the previous node (prev).
4. If prev is not NULL, update its next pointer to NULL.
5. If prev is NULL, set head to NULL since the list had only one node.
6. Delete the original last node (temp).

**8. Reverse the Doubly Linked List (reverse\_list method):**

**Algorithm:**

1. Initialization:

* current: Pointer to the current node being processed, initially set to the head of the list.
* temp: Temporary pointer to facilitate swapping of prev and next pointers.

1. Traversal and Reversal:

* Enter a loop that continues until the end of the list (current != NULL).
* Inside the loop:
* Save the current node's original prev pointer in temp.
* Swap the prev and next pointers of the current node to reverse the linkage.
* Move to the next node in the original order (current = current->prev).

1. Update Head:

* After the loop, if the list was not empty (temp != NULL), update the head pointer to point to the new first node in the reversed list.