**1. Algorithm for create()**

**Purpose**: Creates a doubly linked list with n nodes.

**Algorithm**:

1. Read the number of nodes n.
2. Initialize a loop from i = 0 to n - 1:
   * Allocate memory for a new node.
   * Read data into the node.
   * If head is NULL (list is empty):
     + Set head = tail = new.
   * Else:
     + Set tail->next = new.
     + Set new->prev = tail.
     + Update tail = new.
     + Set tail->next = NULL.

**2. Algorithm for display()**

**Purpose**: Displays the elements of the doubly linked list.

**Algorithm**:

1. If head is NULL, print "List is empty" and return.
2. Initialize temp = head.
3. While temp is not NULL:
   * Print temp->data.
   * Move temp = temp->next.

**3. Algorithm for insert()**

**Purpose**: Calls appropriate insertion function based on user's choice.

**Algorithm**:

1. Read the user's choice c.
2. If c == 1, call insertatbeginning().
3. If c == 2, call insertafteranynode().
4. If c == 3, call insertatend().

**4. Algorithm for insertatbeginning()**

**Purpose**: Inserts a new node at the beginning of the list.

**Algorithm**:

1. Allocate memory for a new node.
2. Read data into new.
3. If head is NULL:
   * Set head = tail = new.
4. Else:
   * Set new->next = head.
   * Set head->prev = new.
   * Set new->prev = NULL.
   * Update head = new.

**5. Algorithm for insertatend()**

**Purpose**: Inserts a new node at the end of the list.

**Algorithm**:

1. Allocate memory for a new node.
2. Read data into new.
3. If head is NULL:
   * Set head = tail = new.
4. Else:
   * Set tail->next = new.
   * Set new->prev = tail.
   * Update tail = new.
   * Set tail->next = NULL.

**6. Algorithm for insertafteranynode()**

**Purpose**: Inserts a new node after a specified node.

**Algorithm**:

1. Read the value q after which to insert.
2. Initialize temp = head.
3. Traverse the list until temp->data == q or temp == NULL.
4. If temp == NULL, print "Node not found" and return.
5. Allocate memory for new and read data into it.
6. Set new->next = temp->next.
7. If temp->next != NULL, set temp->next->prev = new.
8. Set temp->next = new.
9. Set new->prev = temp.

**7. Algorithm for delete()**

**Purpose**: Calls appropriate deletion function based on user's choice.

**Algorithm**:

1. Read the user's choice h.
2. If h == 1, call deleteatbeginning().
3. If h == 2, call deleteafteranynode().
4. If h == 3, call deleteatend().

**8. Algorithm for deleteatbeginning()**

**Purpose**: Deletes the first node in the list.

**Algorithm**:

1. If head == NULL, print "No data to delete" and return.
2. Store head in p.
3. Update head = head->next.
4. If head != NULL, set head->prev = NULL; otherwise, set tail = NULL.
5. Free p.

**9. Algorithm for deleteatend()**

**Purpose**: Deletes the last node in the list.

**Algorithm**:

1. If head == NULL, print "No data to delete" and return.
2. If head == tail, free head, set head = tail = NULL, and return.
3. Initialize temp = head.
4. Traverse the list until temp->next == tail.
5. Free tail.
6. Update tail = temp.
7. Set tail->next = NULL.

**10. Algorithm for deleteafteranynode()**

**Purpose**: Deletes a node after a specified node.

**Algorithm**:

1. Read the value p after which to delete.
2. Initialize temp = head.
3. Traverse the list until temp->data == p or temp == NULL.
4. If temp == NULL or temp->next == NULL, print "No node to delete" and return.
5. Store temp->next in q.
6. Set temp->next = q->next.
7. If q->next != NULL, set q->next->prev = temp.
8. Free q.