**1️ Algorithm for create()**

1. **Input**: Number of nodes, node data.
2. **Steps**:
   * Prompt the user for the number of nodes.
   * For each node:
     + Create a new node using malloc.
     + Input the data.
     + If head is NULL, set head and tail to the new node.
     + Else, update tail->next to the new node and move tail to the new node.
   * End with tail->next = NULL.
3. **Output**: A linked list of n nodes.

**2️ Algorithm for display()**

1. **Input**: Linked list.
2. **Steps**:
   * If head is NULL, print "List is empty".
   * Otherwise, traverse from head to NULL.
     + Print data of each node.
   * Stop when temp == NULL.
3. **Output**: Print all node data.

**3️ Algorithm for insert()**

1. **Input**: Insertion choice (beginning, after a node, end).
2. **Steps**:
   * Take user choice.
   * Call the corresponding insertion function:
     + insertatbeginning(), insertafteranynode(), or insertatend().
3. **Output**: Insertion at the specified location.

**4️ Algorithm for insertatbeginning()**

1. **Input**: New node data.
2. **Steps**:
   * Create a new node.
   * Set new->data to user input.
   * If the list is empty, set head = tail = new.
   * Else:
     + Set new->next = head.
     + Update head = new.
3. **Output**: Node inserted at the beginning.

**5️ Algorithm for insertatend()**

1. **Input**: New node data.
2. **Steps**:
   * Create a new node.
   * Set new->data to user input.
   * If the list is empty, set head = tail = new.
   * Else:
     + Set tail->next = new.
     + Update tail = new.
   * Set tail->next = NULL.
3. **Output**: Node inserted at the end.

**6️ Algorithm for insertafteranynode()**

1. **Input**: Target node data, new node data.
2. **Steps**:
   * Traverse the list to find the node with the given target data.
   * If found:
     + Create a new node.
     + Set new->data with user input.
     + Set new->next = temp->next.
     + Set temp->next = new.
   * If not found, print "Node not found".
3. **Output**: Node inserted after the specified node.

**7️ Algorithm for delete()**

1. **Input**: Deletion choice (beginning, after a node, end).
2. **Steps**:
   * Take user choice.
   * Call the corresponding deletion function:
     + deleteatbeginning(), deleteafteranynode(), or deleteatend().
3. **Output**: Deletion at the specified location.

**8️ Algorithm for deleteatbeginning()**

1. **Input**: Linked list.
2. **Steps**:
   * If the list is empty, print "Underflow".
   * Else:
     + Set p = head.
     + Update head = head->next.
     + Free p.
3. **Output**: First node deleted.

**9️ Algorithm for deleteatend()**

1. **Input**: Linked list.
2. **Steps**:
   * If the list is empty, print "Underflow".
   * Else:
     + Traverse the list until temp->next->next == NULL.
     + Free the last node.
     + Set temp->next = NULL.
3. **Output**: Last node deleted.

**🔟 Algorithm for deleteafteranynode()**

1. **Input**: Target node data.
2. **Steps**:
   * Traverse the list to find the node with the target data.
   * If found:
     + Set q = temp->next.
     + Update temp->next = q->next.
     + Free q.
   * If not found, print "Node not found".
3. **Output**: Node deleted after the specified node.