**1. create() - Creating a Circular Linked List**

**Algorithm:**

1. Take input for the number of nodes to be created.
2. Repeat the following steps for n times:
   1. Allocate memory for a new node.
   2. Take input for the data of the new node.
   3. If the list is empty:
      * Set head = tail = new.
      * Point tail->next to head.
   4. Otherwise:
      * Set tail->next = new.
      * Move tail to the new node.
      * Set tail->next = head.

**2. display() - Displaying the Circular Linked List**

**Algorithm:**

1. Initialize a temporary pointer temp = head.
2. Print "Linked list: ".
3. Traverse the circular linked list:
   * Print temp->data.
   * Move to the next node (temp = temp->next).
   * Stop when temp == head.

**3. insert() - Menu for Insertion**

**Algorithm:**

1. Take input for the insertion choice.
2. Based on the choice:
   * Call insertatbeginning() if inserting at the beginning.
   * Call insertafteranynode() if inserting after a specific node.
   * Call insertatend() if inserting at the end.

**4. insertatbeginning() - Insert at the Beginning**

**Algorithm:**

1. Allocate memory for a new node.
2. Take input for new->data.
3. If the list is empty:
   * Set head = tail = new.
   * Set tail->next = head.
4. Otherwise:
   * Set new->next = head.
   * Update head = new.
   * Set tail->next = head.

**5. insertafteranynode() - Insert After a Specific Node**

**Algorithm:**

1. Take input for the node value p after which insertion is required.
2. Initialize a pointer temp = head.
3. Traverse the list until temp->data == p or temp == head.
4. Allocate memory for a new node.
5. Take input for new->data.
6. Set new->next = temp->next.
7. Set temp->next = new.

**6. insertatend() - Insert at the End**

**Algorithm:**

1. Allocate memory for a new node.
2. Take input for new->data.
3. If the list is empty:
   * Set head = tail = new.
   * Set tail->next = head.
4. Otherwise:
   * Set new->next = head.
   * Set tail->next = new.
   * Move tail to new.

**7. delete() - Menu for Deletion**

**Algorithm:**

1. Take input for the deletion choice.
2. Based on the choice:
   * Call deleteatbeginning() if deleting the first node.
   * Call deleteafteranynode() if deleting a node after a specific node.
   * Call deleteatend() if deleting the last node.

**8. deleteatbeginning() - Delete the First Node**

**Algorithm:**

1. If the list is empty, print "No data to be deleted" and return.
2. Store head in p.
3. Move head to head->next.
4. Free the memory of p.
5. Update tail->next = head.

**9. deleteafteranynode() - Delete a Node After a Specific Node**

**Algorithm:**

1. Take input for the node value p before the node to be deleted.
2. Initialize temp = head.
3. Traverse the list until temp->data == p or temp->next == head.
4. Store temp->next in q.
5. Set temp->next = q->next.
6. Free q.

**10. deleteatend() - Delete the Last Node**

**Algorithm:**

1. If the list is empty, print "No data to be deleted" and return.
2. If the list has only one node:
   * Free head.
   * Set head = tail = NULL.
   * Return.
3. Initialize temp = head.
4. Traverse the list until temp->next->next == head.
5. Store temp->next in last.
6. Set temp->next = head.
7. Free last.
8. Update tail = temp.