Circular Singly Linked List Operations

---

1. Node Structure

```
Node:
    val : integer
    next : pointer to Node
```

---

2. Create Node

Algorithm `create_node()` **Input:** None
Output: Pointer to newly created Node

```
1. Allocate memory for a new node, call it 'current'
2. If memory allocation fails, print "Memory is full" and exit
3. Read value 'val' from user
4. Set current->val = val
5. Set current->next = current  // circular property
6. Return pointer 'current'
```

Time Complexity: O(1)

---

3. Insert at Back

Algorithm `insert_back(end)` **Input:** Pointer to end node
Output: Updated end node

```
1. temp = create_node()
2. If end is NULL:          // empty list
       end = temp
3. Else:                     // list not empty
       temp->next = end->next   // point new node to first node
       end->next = temp         // link last node to new node
       end = temp               // update end pointer
4. Return end
```

Time Complexity: O(1)

---

4. Insert at Front

Algorithm `insert_front(end)` **Input:** Pointer to end node
Output: Updated end node

```
1. front = create_node()
2. If end is NULL:          // empty list
       end = front
3. Else:
       front->next = end->next   // new node points to first node
       end->next = front         // last node links to new first node
4. Return end
```

Time Complexity: O(1)

---

5. Display All Nodes

Algorithm `display(end)` **Input:** Pointer to end node
Output: Print all nodes

```
1. If end == NULL:
       Print "CIRCULAR LINKED LIST IS EMPTY"
       Return
2. front = end->next  // first node
3. Do:
       Print front->val
       front = front->next
   While front != end->next
4. Print newline
```

Time Complexity: O(n)

---

6. Delete Front Node

Algorithm `delete_front(end)` **Input:** Pointer to end node
Output: Updated end node

```
1. If end == NULL:
       Print "CIRCULAR LINKED LIST IS EMPTY"
       Return NULL
2. temp = end->next  // first node
3. If end == end->next:  // only one node
       Free end
       Return NULL
```

```
4. Else:
       end->next = temp->next  // remove first node
       Free temp
5. Return end
```

Time Complexity: O(1)

---

7. Delete Back Node

Algorithm `delete_back(end)` **Input:** Pointer to end node
Output: Updated end node

```
1. If end == NULL:
       Print "CIRCULAR LINKED LIST IS EMPTY"
       Return NULL
2. If end == end->next:  // only one node
       Free end
       Return NULL
3. temp = end->next      // first node
4. While temp->next != end:  // traverse to second last node
       temp = temp->next
5. temp->next = end->next  // maintain circular link
6. Free end
7. end = temp             // update end pointer
8. Return end
```

Time Complexity: O(n)

---

8. Main Menu Algorithm

```
1. Initialize end = NULL
2. Loop infinitely:
    a. Print menu:
       1) Insert from Back
       2) Insert from Front
       3) Delete from Front
       4) Delete from Back
       5) Display
       6) Exit
    b. Read choice 'cho' from user
    c. Switch cho:
        Case 1: end = insert_back(end)
        Case 2: end = insert_front(end)
        Case 3: end = delete_front(end)
        Case 4: end = delete_back(end)
```

```
        Case 5: display(end)
        Case 6: Print "Exiting.." and exit
        Default: Print "Please enter correct input"
```

Time Complexity: Depends on operation selected (O(1) for insert/delete, O(n) for display)