Circular Linked List Implementation in C++

Objective:
The objective of this task is to understand and implement a Circular Linked List using C++. This task focuses on pointer manipulation, dynamic memory allocation, and traversal of a list where the last node points back to the first node.

Problem Statement

Implement a Circular Linked List with the following operations:
- Insert nodes at the end of the list
- Display all elements of the circular linked list

Concepts Used

- Circular Linked List
- Dynamic Memory Allocation
- Pointers
- Classes and Objects
- Traversal Techniques

Source Code (C++)

```
#include <iostream>
using namespace std;

class Node {
public:
        int data;
        Node* next;

        Node(int value) {
        data = value;
        next = NULL;
        }
};
```

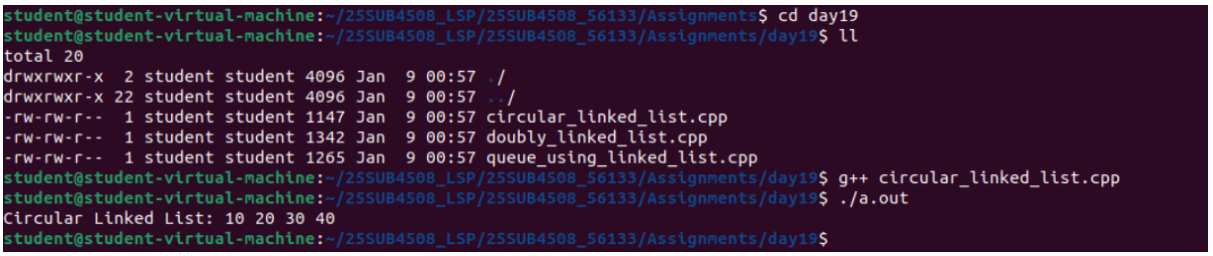
```
class CircularLinkedList {
private:
        Node* head;

public:
        CircularLinkedList() {
        head = NULL;
        }

        // Insert node at the end
        void insertEnd(int value) {
        Node* newNode = new Node(value);

        if (head == NULL) {
        head = newNode;
        newNode->next = head;
        return;
        }

       Node* temp = head;
        while (temp->next != head) {
        temp = temp->next;
        }

        temp->next = newNode;
        newNode->next = head;
        }

        // Display the list
        void display() {
        if (head == NULL) {
        cout << "List is empty." << endl;
        return;
        }

        Node* temp = head;
        cout << "Circular Linked List: ";
        do {
        cout << temp->data << " ";
        temp = temp->next;
        } while (temp != head);
          cout << endl;
        }
};

int main() {
        CircularLinkedList cll;
```

```
    cll.insertEnd(10);
    cll.insertEnd(20);
    cll.insertEnd(30);
    cll.insertEnd(40);

    cll.display();

    return 0;
}
```

Explanation of the Code

1. The Node class represents each node containing data and a pointer to the next node.
2. The CircularLinkedList class maintains a head pointer.
3. In insertEnd(), the last node's next pointer is linked back to the head.
4. If the list is empty, the new node becomes the head and points to itself.
5. The display() function uses a do-while loop to traverse the circular list.
6. Traversal stops when the pointer returns to the head node.
7. The main() function demonstrates insertion and traversal operations.

Program Output



Conclusion

This task demonstrates the implementation of a Circular Linked List in C++. In a circular linked list, the last node connects back to the first node, forming a loop. Such data structures are useful in applications like CPU scheduling, buffering systems, and real-time data processing.