Day 121/180 Circular LinkedList Problems

1: Circular Linked List Traversal

The provided C++ function `printList` prints the data of each node in a circular linked list starting from the given `head`. It uses a pointer `p` to traverse the list, printing the data of each node inside a loop. The loop continues until it reaches the last node, and then it prints the data of the last node outside the loop. The condition `**p->next != head**` ensures that the loop stops when it completes one full traversal of the circular linked list.

```
void printList(struct Node *head)
{
  // code here

       Node*p=head;

       while(p->next!=head){
         cout<<p->data<<" ";
           p=p->next;
       }
      cout<<p->data;

}
```

2: Check If Circular Linked List

The C++ function `isCircular` checks whether a linked list is circular. It uses a pointer to traverse the list, checking if the next node is not `NULL` and not equal to the head. If the loop completes without finding a circular structure, it returns `false`; otherwise, it returns `true`. The final `if-else` statement determines the result based on the state of the last node.

```
bool isCircular(Node *head)
{
   // Your code here
       Node*p=head;

        while(p->next != NULL && p->next != head)
        {
            p = p->next;
        }

        if(p->next == NULL)
        {
            return false;
        }
        else
        {
            return true;
        }


}
```

3: Split a Circular Linked List into two halves

The `splitList` function uses the concept of slow and fast pointers to divide a circular linked list into two halves. It initializes three pointers (`fast`, `slow`, and `temp`) and traverses the list. The `fast` pointer advances two nodes at a time, the `slow` pointer advances one node at a time. When the `fast` pointer reaches the end (or the second-to-last node), it sets `temp` to the current `fast` position.

The midpoint is identified, and the head pointers of two separate linked lists (`head1_ref` and `head2_ref`) are updated. Finally, the circular linkage is broken by updating `temp->next` and `slow->next`. The result is two circular linked lists, each starting from the original head and the node following the midpoint, respectively.

```
void splitList(Node *head, Node **head1_ref, Node **head2_ref)
{
    // your code goes here
    if(!head || !head->next){
        *head1_ref = head;
        *head2_ref = NULL;

        return;
    }

    Node *fast = head;
    Node *slow = head;
    Node *temp = head;

    while(true){

        if(fast->next == head){
            temp = fast;
            break;
        }
        if(fast->next->next == head){
            temp = fast->next;
            break;
        }

        fast = fast->next->next;
        slow = slow->next;
    }

    *head1_ref = head;
    *head2_ref = slow->next;

    temp->next = *head2_ref;
    slow->next = *head1_ref;

    return;
}
```

4: Sorted insert for circular linked list

The `sortedInsert` function in C++ inserts a new node with a given data value into a sorted circular linked list while maintaining the sorted order. It creates a new node and finds the correct position to insert it by traversing the list. The function handles cases where the list is initially empty or when the new node needs to be inserted before the current head. After insertion, it returns the head of the modified circular linked list.

```
 Node *sortedInsert(Node* head, int data)
    {
        //Your code here
         //Your code here
        Node *temp=new Node(data);

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

        Node *curr=head;
        // runs loop until the data is lesser than curr node or curr
points to head node
        while(curr->next!=head && temp->data>curr->data){
            curr=curr->next;
        }
        // add the new node
        temp->next=curr->next;
        curr->next=temp;


        //if lesser than the last node but loop end due to curr->next
=head
        if(curr->data > temp->data){
            int t=curr->data;
```

```
            curr->data=temp->data;
            temp->data=t;
        }

        return head;
    }
```

5: Create a Circular Singly Linked List and also a Circular Doubly Linked List from the given array below.

Int Arr[] = [1,2,3,4,5,6]

Circular Singly Linked List:

```
#include <iostream>

using namespace std;

// Node structure for Circular Singly Linked List
struct Node {
    int data;
    Node* next;
};

// Function to create a Circular Singly Linked List from an array
Node* createCircularSinglyLinkedList(int arr[], int n) {
    if (n == 0) {
        return nullptr;
    }

    Node* head = new Node{arr[0], nullptr};
```

```
    Node* tail = head;

    for (int i = 1; i < n; ++i) {
        tail->next = new Node{arr[i], nullptr};
        tail = tail->next;
    }

    // Make the list circular
    tail->next = head;

    return head;
}

// Function to print the Circular Singly Linked List
void printCircularSinglyLinkedList(Node* head) {
    if (!head) {
        return;
    }

    Node* current = head;
    do {
        cout << current->data << " ";
        current = current->next;
    } while (current != head);

    cout << endl;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    Node* circularSinglyLinkedList =
createCircularSinglyLinkedList(arr, n);

    // Print the Circular Singly Linked List
    printCircularSinglyLinkedList(circularSinglyLinkedList);

    return 0;
```

```
}
```

Circular Doubly Linked List:

```
#include <iostream>

using namespace std;

// Node structure for Circular Doubly Linked List
struct Node {
    int data;
    Node* next;
    Node* prev;
};

// Function to create a Circular Doubly Linked List from an array
Node* createCircularDoublyLinkedList(int arr[], int n) {
    if (n == 0) {
        return nullptr;
    }

    Node* head = new Node{arr[0], nullptr, nullptr};
    Node* tail = head;

    for (int i = 1; i < n; ++i) {
        Node* newNode = new Node{arr[i], nullptr, tail};
        tail->next = newNode;
        tail = newNode;
    }

    // Make the list circular
    tail->next = head;
    head->prev = tail;

    return head;
}
```

```
// Function to print the Circular Doubly Linked List
void printCircularDoublyLinkedList(Node* head) {
    if (!head) {
        return;
    }

    Node* current = head;
    do {
        cout << current->data << " ";
        current = current->next;
    } while (current != head);

    cout << endl;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    Node* circularDoublyLinkedList =
createCircularDoublyLinkedList(arr, n);

    // Print the Circular Doubly Linked List
    printCircularDoublyLinkedList(circularDoublyLinkedList);

    return 0;
}
```