Circular Linked List Basic Operation

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

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

    Node(int data) {
        this->data = data;
        this->next = nullptr;
    }
};

// Insert at Head
void InsertatHead(Node*& head, int data) {
    Node* newnode = new Node(data);
    if (head == nullptr) {
        head = newnode;
        head->next = newnode;  // circular link
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    newnode->next = head;
    temp->next = newnode;
    head = newnode; // update head
}

// Insert at Tail
void Insertattail(Node*& head, int data) {
    Node* newnode = new Node(data);
    if (head == nullptr) {
        head = newnode;
```

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

// Insert at Nth Position
void insertatnthposition(Node*& head, int data, int target) {
    if (target <= 0) {
        cout << "Invalid position" << endl;
        return;
    }
    if (target == 1) {
        InsertatHead(head, data);
        return;
    }
    Node* temp = head;
    for (int i = 1; i < target - 1; i++) {
        temp = temp->next;
        if (temp == head) { // out of range
            cout << "Position out of range" << endl;
            return;
        }
    }
    Node* newnode = new Node(data);
    newnode->next = temp->next;
    temp->next = newnode;
}

// Delete from Head
void deletefromhead(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
```

```
    if (head->next == head) { // only one node
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    Node* todelete = head;
    temp->next = head->next;
    head = head->next;
    delete todelete;
}

// Delete from Tail
void deletefromTail(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    if (head->next == head) { // only one node
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next->next != head) {
        temp = temp->next;
    }
    Node* todelete = temp->next;
    temp->next = head;
    delete todelete;
}

// Delete from Nth Position
void deleteFromNthPosition(Node*& head, int target) {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
```

```
    }
    if (target <= 0) {
        cout << "Invalid position" << endl;
        return;
    }
    if (target == 1) {
        deletefromhead(head);
        return;
    }
    Node* temp = head;
    for (int i = 1; i < target - 1; i++) {
        temp = temp->next;
        if (temp == head) {
            cout << "Position out of range" << endl;
            return;
        }
    }
    Node* todelete = temp->next;
    temp->next = todelete->next;
    delete todelete;
}

// Display list
void Display(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

// Main function
int main() {
    Node* head = nullptr;
```

```
    // Insert at head
    InsertatHead(head, 30);
    InsertatHead(head, 50);
    InsertatHead(head, 28);
    InsertatHead(head, 38);
    InsertatHead(head, 48);

    Display(head);

    // Insert at tail
    Insertattail(head, 70);
    Display(head);

    // Delete from head
    deletefromhead(head);
    Display(head);

    // Delete from tail
    deletefromTail(head);
    Display(head);

    // Insert at position
    insertatnthposition(head, 23, 2);
    Display(head);

    // Delete from nth position
    deleteFromNthPosition(head, 3);
    Display(head);

    return 0;
}
```