Doubly Linked List Basic Operations

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

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

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

// Insert at head
void Insertathead(Node* &head, int data) {
    Node* newnode = new Node(data);
    if (head != nullptr) {
        newnode->next = head;
        head->prev = newnode;
    }
    head = newnode;
}

// Insert at tail
void InsertAttail(Node* &head, int data) {
    Node* newnode = new Node(data);
    if (head == nullptr) {
        head = newnode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
```

```
    }
    temp->next = newnode;
    newnode->prev = temp;
}

// Insert at nth position (1-based index) using for loop and `i` as iterator
void InsertAtPosition(Node* &head, int data, int pos) {
    if (pos <= 0) {
        cout << "Invalid position!" << endl;
        return;
    }

    if (pos == 1) {
        Insertathead(head, data);
        return;
    }

    Node* temp = head;
    for (int i = 1; temp != nullptr && i < pos - 1; i++) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Position out of bounds!" << endl;
        return;
    }

    Node* newnode = new Node(data);
    newnode->next = temp->next;
    newnode->prev = temp;

    if (temp->next != nullptr)
        temp->next->prev = newnode;

    temp->next = newnode;
}

// Delete at head
void deleteathead(Node* &head) {
    if (head == nullptr) {
```

```
        cout << "Linked list is empty" << endl;
        return;
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr)
        head->prev = nullptr;
    delete temp;
}

// Delete at tail
void deleteAttail(Node* &head) {
    if (head == nullptr) {
        cout << "Linked list is empty" << endl;
        return;
    }
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }

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

// Delete at nth position (1-based index) using for loop and `i` as iterator
void DeleteAtPosition(Node* &head, int pos) {
    if (head == nullptr || pos <= 0) {
        cout << "Invalid position or empty list!" << endl;
        return;
    }

    if (pos == 1) {
        deleteathead(head);
        return;
```

```
    }

    Node* temp = head;
    for (int i = 1; temp != nullptr && i < pos; i++) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Position out of bounds!" << endl;
        return;
    }

    if (temp->prev != nullptr)
        temp->prev->next = temp->next;
    if (temp->next != nullptr)
        temp->next->prev = temp->prev;

    delete temp;
}

// Display list
void display(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << "->";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

// Count nodes
int countNodes(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}
```

```
// Sum of nodes
int sumOfNodes(Node* head) {
    int sum = 0;
    Node* temp = head;
    while (temp != nullptr) {
        sum += temp->data;
        temp = temp->next;
    }
    return sum;
}

// Main
int main() {
    Node* head = new Node(20);
    cout<<"Insert at Head"<<endl;
    Insertathead(head, 40);
    Insertathead(head, 11);
    Insertathead(head, 50);
    display(head);

    cout << "Insert at tail:" << endl;
    InsertAttail(head, 60);
    display(head);

    cout << "Insert at position 3 (value 99):" << endl;
    InsertAtPosition(head, 99, 3);
    display(head);

    cout << "Delete at position 4:" << endl;
    DeleteAtPosition(head, 4);
    display(head);

    cout << "Delete at head:" << endl;
    deleteathead(head);
    display(head);

    cout << "Delete at tail:" << endl;
    deleteAttail(head);
    display(head);
```

```
cout << "Number of nodes: " << countNodes(head) << endl;
cout << "Sum of all node values: " << sumOfNodes(head) << endl;

return 0; }
```

Output-

```
Insert at Head
50->11->40->20->NULL
Insert at tail:
50->11->40->20->60->NULL
Insert at position 3 (value 99):
50->11->99->40->20->60->NULL
Delete at position 4:
50->11->99->20->60->NULL
Delete at head:
11->99->20->60->NULL
Delete at tail:
11->99->20->NULL
Number of nodes: 3
Sum of all node values: 130
```