CS-221-L Data Structures and Algorithms Lab

Lab report # 2

Submitted by: Shayan Rizwan Yazdanie

Registration Number: 2024585

Submitted to: Adnan Haider

Semester: 3rd

Faculty of Computer Science and Engineering

GIK Institute of Engineering Sciences and Technology

Task# 01: **Reverse a Double Linked List**

Write a function to reverse a double linked list in-place (without creating a new list).

Example:

Input List: NULL -> 10 -> 20 -> 30 -> 40 -> NULL

Output List: NULL -> 40 -> 30 -> 20 -> 10 -> NULL

Solution

CODE:

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

// NAME: Shayan Rizwan Yazdanie
// REG #: 2024585

// Node structure for doubly linked list
struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int val) {
        data = val;
        next = nullptr;
        prev = nullptr;
    }
};

// Function to reverse a doubly linked list
Node* reverseDoublyLinkedList(Node* head) {
    if (head == nullptr || head->next == nullptr) {
        return head;
    }

    Node* current = head;
    Node* temp = nullptr;

    // Swap next and prev pointers for all nodes
    while (current != nullptr) {
        // Swap next and prev pointers
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;

        // Move to the next node (which is now in prev due to swap)
        current = current->prev;
    }

    // temp is now pointing to the second last node
    // temp->prev is the new head after reversal
    if (temp != nullptr) {
        return temp->prev;
    }

    return head;
}

// Function to print the doubly linked list
void printList(Node* head) {
    cout << "NULL";
    for (Node *ptr = head; ptr != NULL; ptr = ptr->next) {
        cout << "->" << ptr->data;
    }
    cout << "->NULL" << endl;
}

int main() {

    Node *head = new Node(10);
    Node* second = new Node(20);
    Node* third = new Node(30);
    Node* fourth = new Node(40);

    head->next = second;
    second->prev = head;
    second->next = third;
    third->prev = second;
    third->next = fourth;
    fourth->prev = third;

    printList(head);

    head = reverseDoublyLinkedList(head);

    printList(head);



    return 0;
}
```

Task# 02: **Add the Middle Element**

Write a function that finds and add the middle element of a (double) linked list. Example:

Input List: NULL -> 10 -> 20 -> 30 -> 40 -> 50 -> 60 -> NULL

Add node at the middle : 35.

Output List: NULL -> 10 -> 20 -> 30 ->35-> 40 -> 50 -> 60 -> NULL

Solution

CODE:

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

// NAME: Shayan Rizwan Yazdanie
// REG #: 2024585

// For finding the middle element in a doubly linked list, we use the the tortoise and hare
algorithm (slow and fast pointers):
// When the fast pointer cannot move two steps forward (reaches end), the slow pointer will be
at:
// - The middle node for odd-length lists
// - The first middle node for even-length lists

// Insertion happens AFTER slow pointer

// Node structure for doubly linked list
struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int val) {
        data = val;
        next = nullptr;
        prev = nullptr;
    }
};

// Function to add node at the middle of doubly linked list
void addAtMiddle(Node* head, int newData) {
    if (head == nullptr) {
        // If list is empty, create new node as head
        head = new Node(newData);
        return;
    }

    // Find the middle using slow and fast pointers
    Node* slow = head;
    Node* fast = head;

    while (fast != nullptr && fast->next != nullptr && fast->next->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // Now slow points to the node before the insertion point
    // Create new node
    Node* newNode = new Node(newData);

    // Insert after slow pointer
    newNode->next = slow->next;
    newNode->prev = slow;

    if (slow->next != nullptr) {
        slow->next->prev = newNode;
    }
    slow->next = newNode;
}

// Function to print the doubly linked list
void printList(Node* head) {
    cout << "NULL";
    for (Node *ptr = head; ptr != NULL; ptr = ptr->next) {
        cout << "->" << ptr->data;

    }
    cout << "->NULL" << endl;
}

int main() {

    Node* head = new Node(10);
    Node* second = new Node(20);
    Node* third = new Node(30);
    Node* fourth = new Node(40);
    Node* fifth = new Node(50);
    Node* sixth = new Node(60);

    head->next = second;
    second->prev = head;
    second->next = third;
    third->prev = second;
    third->next = fourth;
    fourth->prev = third;
    fourth->next = fifth;
    fifth->prev = fourth;
    fifth->next = sixth;
    sixth->prev = fifth;

    cout << "Original List: ";
    printList(head);

    // Add 35 at the middle
    addAtMiddle(head, 35);


    cout << "After adding 35 at middle: ";
    printList(head);



    return 0;
}
```