DSA Assignment 5

Q1.

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

struct Node {
    int data;
    Node* next;
    Node(int val) { data = val; next = NULL; }
};

class LinkedList {
    Node* head;
public:
    LinkedList() { head = NULL; }

    void insertAtBeginning(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    void insertAtEnd(int val) {
        Node* newNode = new Node(val);
        if (!head) { head = newNode; return; }
        Node* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }

    void insertAfter(int key, int val) {
        Node* temp = head;
        while (temp && temp->data != key) temp = temp->next;
        if (temp) {
            Node* newNode = new Node(val);
            newNode->next = temp->next;
            temp->next = newNode;
        } else cout << "Key not found\n";
    }

    void insertBefore(int key, int val) {
        if (!head) { cout << "List empty\n"; return; }
        if (head->data == key) { insertAtBeginning(val); return; }
        Node* temp = head;
        while (temp->next && temp->next->data != key) temp = temp->next;
        if (temp->next) {
            Node* newNode = new Node(val);
            newNode->next = temp->next;
            temp->next = newNode;
        } else cout << "Key not found\n";
    }

    void deleteFromBeginning() {
        if (!head) { cout << "List empty\n"; return; }
        Node* temp = head;
        head = head->next;
```

```
        delete temp;
    }

    void deleteFromEnd() {
        if (!head) { cout << "List empty\n"; return; }
        if (!head->next) { delete head; head = NULL; return; }
        Node* temp = head;
        while (temp->next->next) temp = temp->next;
        delete temp->next;
        temp->next = NULL;
    }

    void deleteNode(int key) {
        if (!head) { cout << "List empty\n"; return; }
        if (head->data == key) { deleteFromBeginning(); return; }
        Node* temp = head;
        while (temp->next && temp->next->data != key) temp = temp->next;
        if (temp->next) {
            Node* toDelete = temp->next;
            temp->next = temp->next->next;
            delete toDelete;
        } else cout << "Node not found\n";
    }

    void searchNode(int key) {
        Node* temp = head;
        int pos = 1;
        while (temp) {
            if (temp->data == key) {
                cout << "Node found at position " << pos << endl;
                return;
            }
            temp = temp->next;
            pos++;
        }
        cout << "Node not found\n";
    }

    void display() {
        if (!head) { cout << "List empty\n"; return; }
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << " -> ";
            temp = temp->next;
        }
        cout << " -> NULL\n";
    }
};

int main() {
    LinkedList list;
    int choice, val, key;
    while (true) {
        cout << "\n1-Insert at Beginning \n2-Insert at End \n3-Insert After \n4-
Insert Before\n";
        cout << "5-Delete from Beginning \n6-Delete from End \n7-Delete Node \n8-
Search \n9-Display \n10-Exit\n";
        cin >> choice;
        switch (choice) {
```

```
            case 1: cin >> val; list.insertAtBeginning(val); break;
            case 2: cin >> val; list.insertAtEnd(val); break;
            case 3: cin >> key >> val; list.insertAfter(key, val); break;
            case 4: cin >> key >> val; list.insertBefore(key, val); break;
            case 5: list.deleteFromBeginning(); break;
            case 6: list.deleteFromEnd(); break;
            case 7: cin >> key; list.deleteNode(key); break;
            case 8: cin >> key; list.searchNode(key); break;
            case 9: list.display(); break;
            case 10: return 0;
            default: cout << "Invalid choice\n";
        }
    }
}
```

Q2.

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

struct Node {
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

class LinkedList {
    Node* head;
public:
    LinkedList() { head = NULL; }

    void push_back(int val) {
        Node* newNode = new Node(val);
        if (head == NULL) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != NULL) temp = temp->next;
        temp->next = newNode;
    }

    int countAndDeleteKey(int key) {
        int count = 0;

        // Remove occurrences from the beginning
        while (head != NULL && head->data == key) {
            Node* temp = head;
            head = head->next;
            delete temp;
            count++;
        }

        // Now handle rest of the list
        Node* curr = head;
        while (curr != NULL && curr->next != NULL) {
            if (curr->next->data == key) {
```

```
                Node* temp = curr->next;
                curr->next = curr->next->next;
                delete temp;
                count++;
            } else {
                curr = curr->next;
            }
        }
        return count;
    }

    void display() {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data;
            if (temp->next) cout << " -> ";
            temp = temp->next;
        }
        cout << " -> NULL\n";
    }
};

int main() {
    LinkedList list;
    list.push_back(1);
    list.push_back(2);
    list.push_back(1);
    list.push_back(2);
    list.push_back(1);
    list.push_back(3);
    list.push_back(1);

    cout << "Original List: ";
    list.display();

    int key = 2;
    int count = list.countAndDeleteKey(key);

    cout << "Occurrences of " << key << ": " << count << endl;
    cout << "List after deleting all occurrences of " << key << ": ";
    list.display();

    return 0;
}
```

Q3.

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

struct Node {
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

class LinkedList {
```

```
    Node* head;
public:
    LinkedList() { head = NULL; }

    void push_back(int val) {
        Node* newNode = new Node(val);
        if (head == NULL) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != NULL) temp = temp->next;
        temp->next = newNode;
    }

    void findMiddle() {
        if (head == NULL) {
            cout << "List is empty\n";
            return;
        }
        Node* slow = head;
        Node* fast = head;
        while (fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
        }
        cout << "Middle: " << slow->data << endl;
    }

    void display() {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data;
            if (temp->next) cout << " -> ";
            temp = temp->next;
        }
        cout << " -> NULL\n";
    }
};

int main() {
    LinkedList list;
    list.push_back(1);
    list.push_back(2);
    list.push_back(3);
    list.push_back(4);
    list.push_back(5);

    list.display();
    list.findMiddle();

    return 0;
}
```

Q4.

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

struct Node {
```

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

class LinkedList {
    Node* head;
public:
    LinkedList() { head = NULL; }

    void push_back(int val) {
        Node* newNode = new Node(val);
        if (!head) head = newNode;
        else {
            Node* temp = head;
            while (temp->next) temp = temp->next;
            temp->next = newNode;
        }
    }

    void reverse() {
        Node* prev = NULL;
        Node* curr = head;
        Node* next = NULL;
        while (curr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        head = prev;
    }

    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << " -> ";
            temp = temp->next;
        }
        cout << " -> NULL" << endl;
    }
};

int main() {
    LinkedList list;
    list.push_back(1);
    list.push_back(2);
    list.push_back(3);
    list.push_back(4);

    cout << "Original List: \n";
    list.display();

    list.reverse();

    cout << "Reversed List: \n";
```

```
    list.display();

    return 0;
}
```