DATA STRUCTURES AND ALGORITHMS – ASSIGNMENT 5
LINKED LISTS

AADITA GARG
1024030461 – 2C32

(1) Develop a menu driven program for the following operations on a Singly Linked List.

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

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

// Function to insert at the beginning
Node* insertAtBeginning(Node* head, int val) {
   Node* newNode = new Node(val);
   newNode->next = head;
   return newNode;
}

// Function to insert at the end
Node* insertAtEnd(Node* head, int val) {
   Node* newNode = new Node(val);
   if (head == nullptr) return newNode;

   Node* temp = head;
   while (temp->next != nullptr)
      temp = temp->next;
   temp->next = newNode;
   return head;
}

// Function to insert before or after a specific value
Node* insertInBetween(Node* head, int val, int key, bool after=true) {
   Node* newNode = new Node(val);
   if (head == nullptr) return nullptr;
```

```
    Node* temp = head;

    if (!after && head->data == key) {
        newNode->next = head;
        return newNode;
    }

    while (temp != nullptr) {
        if (after && temp->data == key) {
            newNode->next = temp->next;
            temp->next = newNode;
            return head;
        } else if (!after && temp->next != nullptr && temp->next->data == key) { // insert before
            newNode->next = temp->next;
            temp->next = newNode;
            return head;
        }
        temp = temp->next;
    }

    cout << "Key not found!\n";
    delete newNode;
    return head;
}

// Function to delete from beginning
Node* deleteFromBeginning(Node* head) {
    if (head == nullptr) return nullptr;
    Node* temp = head;
    head = head->next;
    delete temp;
    return head;
}

// Function to delete from end
Node* deleteFromEnd(Node* head) {
    if (head == nullptr) return nullptr;
    if (head->next == nullptr) {
        delete head;
        return nullptr;
    }

    Node* temp = head;
    while (temp->next->next != nullptr)
        temp = temp->next;
    delete temp->next;
```

```
    temp->next = nullptr;
    return head;
}

// Function to delete a specific node
Node* deleteNode(Node* head, int key) {
    if (head == nullptr) return nullptr;

    if (head->data == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return head;
    }

    Node* temp = head;
    while (temp->next != nullptr && temp->next->data != key)
        temp = temp->next;

    if (temp->next == nullptr) {
        cout << "Node not found!\n";
        return head;
    }

    Node* del = temp->next;
    temp->next = temp->next->next;
    delete del;
    return head;
}

// Function to search for a node
void searchNode(Node* head, int key) {
    Node* temp = head;
    int pos = 1;
    while (temp != nullptr) {
        if (temp->data == key) {
            cout << "Node " << key << " found at position " << pos << endl;
            return;
        }
        temp = temp->next;
        pos++;
    }
    cout << "Node not found!\n";
}

// Function to display the linked list
void displayList(Node* head) {
```

```
    if (head == nullptr) {
        cout << "List is empty\n";
        return;
    }
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) cout << "->";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int choice, val, key;
    bool after;

    do {
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter value: ";
                cin >> val;
                head = insertAtBeginning(head, val);
                break;
            case 2:
                cout << "Enter value: ";
                cin >> val;
                head = insertAtEnd(head, val);
                break;
            case 3:
                cout << "Enter value to insert: ";
                cin >> val;
                cout << "Enter key node: ";
                cin >> key;
                cout << "Insert after key? (1 for after, 0 for before): ";
                cin >> after;
                head = insertInBetween(head, val, key, after);
                break;
            case 4:
                head = deleteFromBeginning(head);
                break;
            case 5:
                head = deleteFromEnd(head);
```

```
            break;
        case 6:
            cout << "Enter node to delete: ";
            cin >> key;
            head = deleteNode(head, key);
            break;
        case 7:
            cout << "Enter node to search: ";
            cin >> key;
            searchNode(head, key);
            break;
        case 8:
            displayList(head);
            break;
        case 9:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice!\n";
    }

  } while(choice != 9);

  return 0;
}
```

TERMINAL

```
Enter your choice: 1
Enter value: 1
Enter your choice: 1
Enter value: 2
Enter your choice: 1
Enter value: 3
Enter your choice: 2
Enter value: 4
Enter your choice: 2
Enter value: 5
Enter your choice: 3
Enter value to insert: 6
Enter key node: 5
Insert after key? (1 for after, 0 for before): 0
Enter your choice: 8
3->2->1->4->6->5
Enter your choice: 4
Enter your choice: 8
2->1->4->6->5
Enter your choice: 5
Enter your choice: 8
2->1->4->6
Enter your choice: 6
Enter node to delete: 2
Enter your choice: 8
1->4->6
Enter your choice: 7
Enter node to search: 4
Node 4 found at position 2
Enter your choice: 8
1->4->6
Enter your choice: 9
Exiting...
```

(2) Write a program to count the number of occurrences of a given key in a singly linked list and then delete all the occurrences.

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

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

// Function to count occurrences
int countOccurrences(Node* head, int key) {
   int count = 0;
   Node* curr = head;
   while (curr != nullptr) {
      if (curr->data == key)
         count++;
      curr = curr->next;
   }
   return count;
}

// Function to delete all occurrences of key
Node* deleteOccurrences(Node* head, int key) {
   // Remove leading nodes
   while (head != nullptr && head->data == key) {
      Node* temp = head;
      head = head->next;
      delete temp;
   }

   Node* curr = head;
   Node* prev = nullptr;

   while (curr != nullptr) {
      if (curr->data == key) {
         prev->next = curr->next;
         delete curr;
         curr = prev->next;
      } else {
         prev = curr;
         curr = curr->next;
      }
   }
```

```
    return head;
}

void printList(Node* head) {
    Node* curr = head;
    while (curr != nullptr) {
        cout << curr->data;
        if (curr->next != nullptr) cout << "->";
        curr = curr->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(1);
    head->next->next->next = new Node(2);
    head->next->next->next->next = new Node(1);
    head->next->next->next->next->next = new Node(3);
    head->next->next->next->next->next->next = new Node(1);

    int key = 1;

    int count = countOccurrences(head, key);
    cout << "Count: " << count << endl;

    head = deleteOccurrences(head, key);
    cout << "Updated Linked List: ";
    printList(head);

    return 0;
}
```

```
TERMINAL

Count: 4
Updated Linked List: 2->2->3


** Process exited - Return Code: 0 **
```

(3) Write a program to find the middle of a linked list.

```
#include <iostream>
using namespace std;
// Node definition
struct Node {
   int data;
   Node* next;
   Node(int val) {
      data = val;
      next = nullptr;
   }
};

// Function to find middle node
Node* findMiddle(Node* head) {
   if (head == nullptr) return nullptr;

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

   while (fast != nullptr && fast->next != nullptr) {
      slow = slow->next;       // move 1 step
      fast = fast->next->next;  // move 2 steps
   }
   return slow;
}

int main() {
   Node* head = new Node(1);
   head->next = new Node(2);
   head->next->next = new Node(3);
   head->next->next->next = new Node(4);
   head->next->next->next->next = new Node(5);

   Node* middle = findMiddle(head);
   if (middle != nullptr)
      cout << "Middle element is: " << middle->data << endl;
   return 0;
}
```

```
TERMINAL

Middle element is: 3


** Process exited - Return Code: 0 **
```

(4) Write a program to reverse a linked list.

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

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

Node* reverseList(Node* head) {
   Node* prev = nullptr;
   Node* curr = head;
   Node* next = nullptr;

   while (curr != nullptr) {
      next = curr->next;   // save next node
      curr->next = prev;   // reverse link
      prev = curr;         // move prev forward
      curr = next;         // move curr forward
   }
   return prev; // new head
}

void printList(Node* head) {
   while (head != nullptr) {
      cout << head->data;
      if (head->next != nullptr) cout << "->";  // print arrow only if not last node
      head = head->next;
   }
   cout << endl;
}

int main() {
   Node* head = new Node(1);
   head->next = new Node(2);
   head->next->next = new Node(3);
   head->next->next->next = new Node(4);

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

   head = reverseList(head);

   cout << "Reversed list: ";
```

```
    printList(head);

    return 0;
}
```