Lab Assignment 5 Solutions (Singly Linked List)

Q1. Menu Driven Program for Singly Linked List:

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;

void insertBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
}

void insertEnd(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

void insertAfter(int key, int data) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != key)
        temp = temp->next;
    if (temp == NULL) return;
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp->next;
    temp->next = newNode;
}

void deleteBeginning() {
    if (head == NULL) return;
    struct Node* temp = head;
    head = head->next;
    free(temp);
}

void deleteEnd() {
    if (head == NULL) return;
    if (head->next == NULL) {
        free(head);
        head = NULL;
        return;
    }
    struct Node* temp = head;
    while (temp->next->next != NULL)
        temp = temp->next;
    free(temp->next);
    temp->next = NULL;
}

void deleteNode(int key) {
    struct Node* temp = head;
    struct Node* prev = NULL;
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
```

```
    }
    if (temp == NULL) return;
    if (prev == NULL) head = head->next;
    else prev->next = temp->next;
    free(temp);
}

void search(int key) {
    struct Node* temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            printf("Element found at position %d\n", pos);
            return;
        }
        temp = temp->next;
        pos++;
    }
    printf("Element not found\n");
}

void display() {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, data, key;
    while (1) {
        printf("\n1.Insert at Beginning\n2.Insert at End\n3.Insert After Key\n");
        printf("4.Delete Beginning\n5.Delete End\n6.Delete Key\n");
        printf("7.Search\n8.Display\n9.Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: printf("Enter data: "); scanf("%d", &data); insertBeginning(data); break;
            case 2: printf("Enter data: "); scanf("%d", &data); insertEnd(data); break;
            case 3: printf("Enter key and data: "); scanf("%d %d", &key, &data); insertAfter(key, d
            case 4: deleteBeginning(); break;
            case 5: deleteEnd(); break;
            case 6: printf("Enter key: "); scanf("%d", &key); deleteNode(key); break;
            case 7: printf("Enter key: "); scanf("%d", &key); search(key); break;
            case 8: display(); break;
            case 9: exit(0);
        }
    }
}
```

Q2. Count occurrences of a key and delete all:

```
void countAndDelete(int key) {
    struct Node* temp = head;
    int count = 0;
    while (temp != NULL) {
        if (temp->data == key)
            count++;
        temp = temp->next;
    }
    printf("Occurrences of %d: %d\n", key, count);
    while (head != NULL && head->data == key) {
        struct Node* t = head;
        head = head->next;
        free(t);
    }
    temp = head;
    struct Node* prev = NULL;
    while (temp != NULL) {
        if (temp->data == key) {
            prev->next = temp->next;
            free(temp);
            temp = prev->next;
        } else {
```

```
            temp = prev->next;
        } else {
            prev = temp;
            temp = temp->next;
        }
    }
}
```

Q3. Find Middle:

```
void findMiddle() {
    struct Node* slow = head;
    struct Node* fast = head;
    if (head == NULL) return;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }
    printf("Middle element: %d\n", slow->data);
}
```

Q4. Reverse Linked List:

```
void reverseList() {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}
```

Additional Q1. Reverse Linked List II:

```
// Reverse a linked list II (between positions m and n)
struct Node* reverseBetween(struct Node* head, int m, int n) {
    if (!head) return NULL;
    struct Node dummy;
    dummy.next = head;
    struct Node* prev = &dummy;
    for (int i = 1; i < m; i++)
        prev = prev->next;
    struct Node* const reverseHead = prev;
    prev = reverseHead->next;
    struct Node* curr = prev->next;
    for (int i = m; i < n; i++) {
        prev->next = curr->next;
        curr->next = reverseHead->next;
        reverseHead->next = curr;
        curr = prev->next;
    }
    return dummy.next;
}
```

Additional Q2. Rotate List:

```
// Rotate a linked list by k positions
struct Node* rotateRight(struct Node* head, int k) {
    if (!head || !head->next || k == 0) return head;
    struct Node* temp = head;
    int len = 1;
    while (temp->next) {
        temp = temp->next;
        len++;
    }
    temp->next = head; // circular
    k = k % len;
    int steps = len - k;
    struct Node* newTail = temp;
```

```
    while (steps--) newTail = newTail->next;
    struct Node* newHead = newTail->next;
    newTail->next = NULL;
    return newHead;
}
```

Additional Q3. Add Two Polynomials:

```
// Adding two polynomials using linked list
struct PolyNode {
    int coeff, power;
    struct PolyNode* next;
};

struct PolyNode* addPolynomial(struct PolyNode* p1, struct PolyNode* p2) {
    struct PolyNode* result = NULL;
    struct PolyNode** lastPtrRef = &result;
    while (p1 && p2) {
        struct PolyNode* temp = (struct PolyNode*)malloc(sizeof(struct PolyNode));
        if (p1->power > p2->power) {
            temp->coeff = p1->coeff;
            temp->power = p1->power;
            p1 = p1->next;
        } else if (p2->power > p1->power) {
            temp->coeff = p2->coeff;
            temp->power = p2->power;
            p2 = p2->next;
        } else {
            temp->coeff = p1->coeff + p2->coeff;
            temp->power = p1->power;
            p1 = p1->next; p2 = p2->next;
        }
        temp->next = NULL;
        *lastPtrRef = temp;
        lastPtrRef = &(temp->next);
    }
    while (p1 || p2) {
        struct PolyNode* temp = (struct PolyNode*)malloc(sizeof(struct PolyNode));
        if (p1) { temp->coeff = p1->coeff; temp->power = p1->power; p1 = p1->next; }
        else { temp->coeff = p2->coeff; temp->power = p2->power; p2 = p2->next; }
        temp->next = NULL;
        *lastPtrRef = temp;
        lastPtrRef = &(temp->next);
    }
    return result;
}
```

Additional Q4. Intersection Point of Two Linked Lists:

```
// Intersection point of two linked lists
int getCount(struct Node* head) {
    int count = 0;
    while (head) { count++; head = head->next; }
    return count;
}

int getIntersection(int d, struct Node* head1, struct Node* head2) {
    for (int i = 0; i < d; i++) head1 = head1->next;
    while (head1 && head2) {
        if (head1 == head2) return head1->data;
        head1 = head1->next;
        head2 = head2->next;
    }
    return -1;
}

int findIntersection(struct Node* head1, struct Node* head2) {
    int c1 = getCount(head1);
    int c2 = getCount(head2);
    if (c1 > c2) return getIntersection(c1 - c2, head1, head2);
    else return getIntersection(c2 - c1, head2, head1);
}
```