Aim ↬ To perform basic operations on Linked List.

Objectives ↬

i. Write a program to create a linked list of integers entered by user and perform the following:
   a. Traverse the linked list and print all the elements from first to last.
   b. Insert a new node at the start of the linked list. Print the new linked list.
   c. Insert a new node at the end of the linked list. Print the new linked list.
   d. Insert a new node after a node having a key value. Print the new linked list.

ii. Write a program to create a linked list of integers entered by user and delete a node:
   a. At a particular position entered by the user
   b. After a node having a key value
   c. Before a node having a key value
   d. Having a key value

   After each deletion print the resultant Linked List.

Software Required ↬ Visual Studio Code

Code 1 ↬

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

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}
```

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

void insertAtStart(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
    printf("\nAfter inserting %d at the start:", data);
    traverseList(*head);
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    traverseList(*head);
}

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

```
    if (temp != NULL) {
        struct Node* newNode = createNode(data);
        newNode->next = temp->next;
        temp->next = newNode;
        printf("\nAfter inserting %d after key %d:", data, key);
        traverseList(head);
    } else {
        printf("\nKey %d not found in the list.", key);
    }
}

int main() {
    struct Node* head = NULL;
    int n, data, key;

    printf("\nEnter number of elements for the linked list: ");
    scanf("%d", &n);
    printf("\nEnter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        insertAtEnd(&head, data);
    }

    traverseList(head);

    printf("\nEnter element to insert at the start: ");
    scanf("%d", &data);
    insertAtStart(&head, data);

    printf("\nEnter element to insert at the end: ");
    scanf("%d", &data);
    insertAtEnd(&head, data);

    printf("\nEnter key value to insert after and the element to insert: ");
    scanf("%d %d", &key, &data);
    insertAfterKey(head, key, data);
```

```
    return 0;
}
```

Output ↬

```
Enter number of elements for the linked list: 5

Enter 5 elements:
-2 4 7 -1 6

Linked List: -2 -> NULL

Linked List: -2 -> 4 -> NULL

Linked List: -2 -> 4 -> 7 -> NULL

Linked List: -2 -> 4 -> 7 -> -1 -> NULL

Linked List: -2 -> 4 -> 7 -> -1 -> 6 -> NULL

Linked List: -2 -> 4 -> 7 -> -1 -> 6 -> NULL

Enter element to insert at the start: 21

After inserting 21 at the start:
Linked List: 21 -> -2 -> 4 -> 7 -> -1 -> 6 -> NULL

Enter element to insert at the end: -7

Linked List: 21 -> -2 -> 4 -> 7 -> -1 -> 6 -> -7 -> NULL

Enter key value to insert after and the element to insert: 4 27

After inserting 27 after key 4:
Linked List: 21 -> -2 -> 4 -> 27 -> 7 -> -1 -> 6 -> -7 -> NULL
```

Code 2 ↬

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

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

struct Node* createNode(int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

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

void deleteAtPosition(struct Node** head, int pos) {
    if (*head == NULL || pos < 1) return;
    struct Node* temp = *head;
    if (pos == 1) {
        *head = temp->next;
        free(temp);
    } else {
        for (int i = 1; temp != NULL && i < pos - 1; i++) {
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) return;
        struct Node* next = temp->next->next;
        free(temp->next);
        temp->next = next;
    }
    printf("\nAfter deletion at position %d:", pos);
    traverseList(*head);
}

void deleteAfterKey(struct Node** head, int key) {
    struct Node* temp = *head;
```

```
    while (temp != NULL && temp->data != key) {
        temp = temp->next;
    }
    if (temp != NULL && temp->next != NULL) {
        struct Node* delNode = temp->next;
        temp->next = temp->next->next;
        free(delNode);
        printf("\nAfter deleting after key %d:", key);
        traverseList(*head);
    }
}

void deleteBeforeKey(struct Node** head, int key) {
    if (*head == NULL || (*head)->next == NULL) return;
    struct Node* prev = NULL, * curr = *head, * next = curr->next;
    if (curr->next->data == key) {
        *head = next;
        free(curr);
    } else {
        while (next->next != NULL && next->next->data != key) {
            prev = curr;
            curr = next;
            next = next->next;
        }
        if (next->next != NULL && next->next->data == key) {
            prev->next = next;
            free(curr);
        }
    }
    printf("\nAfter deleting before key %d:", key);
    traverseList(*head);
}

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

```
        temp = temp->next;
    }
    if (temp == NULL) return;
    if (prev == NULL) *head = temp->next;
    else prev->next = temp->next;
    free(temp);
    printf("\nAfter deleting key %d:", key);
    traverseList(*head);
}

int main() {
    struct Node* head = NULL;
    int n, data, pos, key;

    printf("\nEnter number of elements for the linked list: ");
    scanf("%d", &n);
    printf("\nEnter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        struct Node* newNode = createNode(data);
        if (head == NULL) {
            head = newNode;
        } else {
            struct Node* temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    traverseList(head);

    printf("\nEnter the position to delete: ");
    scanf("%d", &pos);
    deleteAtPosition(&head, pos);
```

```
    printf("\nEnter the key value after which to delete: ");
    scanf("%d", &key);
    deleteAfterKey(&head, key);

    printf("\nEnter the key value before which to delete: ");
    scanf("%d", &key);
    deleteBeforeKey(&head, key);

    printf("\nEnter the key value to delete: ");
    scanf("%d", &key);
    deleteKey(&head, key);

    return 0;
}
```

Output ↬

```
Enter number of elements for the linked list: 6

Enter 6 elements:
-4 2 7 21 -25 5

Linked List: -4 -> 2 -> 7 -> 21 -> -25 -> 5 -> NULL

Enter the position to delete: 3

After deletion at position 3:
Linked List: -4 -> 2 -> 21 -> -25 -> 5 -> NULL

Enter the key value after which to delete: -25

After deleting after key -25:
Linked List: -4 -> 2 -> 21 -> -25 -> NULL

Enter the key value before which to delete: 2

After deleting before key 2:
Linked List: 2 -> 21 -> -25 -> NULL

Enter the key value to delete: 21

After deleting key 21:
Linked List: 2 -> -25 -> NULL
```

Result ↬

The programs successfully handled linked list operations, including creation, traversal, insertion, and deletion. They accurately performed the following:

- **Traversal:** Displayed all linked list elements.
- **Insertion:** Added nodes at the start, end, and after a specific key value.
- **Deletion:** Removed nodes by position, after a key value, before a key value, and by key value.

Conclusion ↬

The experiment demonstrated core linked list operations in C, including node insertion, deletion, and traversal. The programs illustrated effective management of linked lists, enhancing understanding of dynamic memory allocation and node manipulation.

Precautions ↬

- Ensure valid inputs for matrix elements and sizes to prevent incorrect results and runtime errors.
- Properly allocate and deallocate memory to avoid memory leaks and access violations.
- Check for cases of empty lists or invalid operations, such as deleting from an empty list or invalid positions.
- Implement checks for successful insertion and deletion, and handle scenarios where nodes or keys are not found.