LAB-5 (Deletion of elements in a Linked List) (18/1/24).

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

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

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

18/1/24

```
void deleteFirst(struct Node **head){
    if(*head != NULL){
        struct Node *temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}

void deleteElement(struct Node **head, int value){
    struct Node *current = *head;
    *head = (*head)->next;
    free(temp);

    struct Node *current = *head;
    struct Node *prev = NULL;

    while(current != NULL && current->data != value){
        prev = current;
        current = current->next;
    }
    if(current == NULL){
        return;
    }
    if(prev == NULL){
        *head = current->next;
    }
    else{
        prev->next = current->next;
    }
    free(current);
}
```

```
void deleteLast(struct Node **head) {
    if (*head == NULL) {
        return;
    }
    struct Node *temp = *head;
    struct Node *prev = NULL;

    while (temp->next != NULL)
    {
        prev = temp;
        temp = temp->next;
    }
    if (prev == NULL) {
        *head = NULL;
    }
    else {
        prev->next = NULL;
    }
    free(temp);
}

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

int main()
{
    struct Node *head = NULL;
```

```
    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);

    printf("Initial Linked List: ");
    displayList(head);

    deleteFirst(&head);
    printf("\nAfter deleting the first element: ");
    displayList(head);

    deleteElement(&head, 2);
    printf("\nAfter deleting specified element (2): ");
    displayList(head);

    deleteLast(&head);
    printf("\nAfter deleting the last element: ");
    displayList(head);

    return 0;
}
```

Output :

Initial Linked List : 1 → 2 → 3 → NULL

After deleting the first element : 2 → 3 → NULL

After deleting specified element (2) : 3 → NULL

After deleting the last element : NULL

Leetcode Pgm

Lab 11/1/24 - pgm completed on 19/1/24

(Minstack Problem) - 18/1/24 - NLP 19/1/24

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

typedef struct
{
    int value;
    int main;
} stackNode;

typedef struct
{
  stackNode *array;
  int capacity;
  int top;
}minstack;

MinStack * minStackCreate()
{
  MinStack * stack = (MinStack *) malloc (sizeof (MinStack));
  Stack → capacity = 10;
  Stack → array = (StackNode *) malloc (stack → capacity *
                  sizeof (StackNode));

  Stack → top = -1;
  return stack;
}
```

```
void minStackPush (MinStack * obj , int val)
{
    if (obj -> top == obj -> capacity - 1)
    {
        obj -> capacity *= 2;
        obj -> array = (StackNode *) realloc (obj -> array,
                        obj -> capacity * sizeof (StackNode));
    }
    StackNode newNode;
    newNode.value = val;
    newNode.min = (obj -> top == -1) ? val : (val < obj ->
                  array [obj -> top].min) ? val : obj ->
                  array [obj -> top].min;
    obj -> array [++(obj -> top)] = newNode;
}

void minStackPop (MinStack * obj)
{
    if (obj -> top != -1)
    {
        obj -> top --;
    }
}

int minStackTop (MinStack * obj)
{
    if (obj -> top != -1)
    {
        return obj -> array [obj -> top].value;
    }
    return -1;
}
```

```
int minStackGetMin (MinStack * obj)
{
    if (obj -> top! = -1)
    {
        return obj -> array [obj -> top].min;
    }
    return -1;
}

void minStackFree (MinStack * obj)
{
    free (obj -> array);
    free (obj);
}
```