Week - 04

18/01/2024

// Singly linked list delete and display implementation

```
#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;
        }
}
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;
    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)
```

ata!

```
    {
    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);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);

    printf("Initial linked list: ");
    displayList(head);

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

    deleteElement(&head, 4);
    printf("\nAfter deleting the last specified
      element(4): ");
    displayList(head);

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

```
    displaylist(head);

    return 0;
}
```

output:-

Initial linked list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
After deleting the first element: 2 -> 3 -> 4 -> 5 -> NULL
After deleting the specified element (4): 2 -> 3 -> 5 -> NULL
After deleting the last element: 2 -> 3 -> NULL.

```
//Leet code 01

typedef struct{
    int size;
    int top;
    int *s;
    int *minstack;
} MinStack;

MinStack* minStackCreate(){
    MinStack *st=(MinStack*)malloc(sizeof(
MinStack));
    if(st==NULL)
    {
        printf("memory allocation failed");
        exit(0);
    }
    st->size=1000;
    st->top=-1;
    st->s=(int*)malloc(st->size*sizeof(int));
    st->minstack=(int*)malloc(st->size*
sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;
}
```

```
void minStackPush(MinStack* obj, int val){
    if (obj->top == obj->Size-1)
    {
        printf("stack is overflow");
    }
    else{
        obj->top++;
        obj->s[obj->top] = val;
        if (obj->top == 0 || val < obj->minstack
[obj->top-1]){
        obj->minstack[obj->top] = val;
        else{
        obj->minstack[obj->top] = obj->minstack[
        obj->top-1];
        }
    }
}
void minStackPop(MinStack* obj){
    int value;
    if (obj->top == -1)
    {
        printf("underflow");
    }
    else
    {
        value = obj->s[obj->top];
        obj->top--;
        printf("%d is popped\n", value);
    }
}
int minStackTop(MinStack* obj){
        int value = -1;
```

```
    if(obj->top==-1)
    {
        printf("underflow\n");
        exit(0);
    }
    else
    {
        value=obj->s[obj->top];
        return value;
    }
}
int minStackGetMin(MinStack* obj){
    if(obj->top==-1)
    {
        printf("underflow\n");
        exit(0);
    }
    else
    {
        return obj->minstack[obj->top];
    }
}
void minStackFree(MinStack* obj){
    free(obj->s);
    free(obj->minstack);
    free(obj);
}
```

output:-

input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

stdout:
-3 is popped
output:
[null, null null, null, -3, null, 0, -2]
Expected-
[null, null, null, null, -3, null, 0, -2]

⇒ //Leet code - 02

```
struct ListNode* reverseBetween(struct ListNode*
head, int left, int right){
    struct ListNode* l=head;
    struct ListNode* r=head;
    int difference = right - left;
    if (left==right){
        return head;
    }
    for(int i=0;i<left-1;i++){
        l = l->next;
    }
    for(int i=0;i<right-1;i++){
        r = r->next;
    }
    printf("%d\n%d\n", l->val, r->val);
    while(difference >= 0){
        int temp = l->val;
        l->val = r->val;
        r->val = temp;
        l = l->next;
        r = l;
        for(int i=0;i<difference-2;i++){
            r = r->next;
        }
    }
```

```
    }
    difference -= 2;
  }
  return head;
}
```

case 1:

Input
head = [1,2,3,4,5]
left = 2
right = 4
stdout
2
4

output
[1,4,3,2,5]

Expected:
[1,4,3,2,5]

case 2:-

Input
head = [5]
left = 1
right = 1
Output
[5]
Expected
[5]