enter the loc 2
deleted at 2 menu
1. delete from beg 2. del from end 3. del at specific pos
4. display 5. exit enter the choice: 5

2 leetcode

```
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=5;
    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) {
    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);
}
```

Leetcode 2.

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
    if(head==NULL || left==right){
        return head;
    }
    struct ListNode* p = malloc(sizeof(struct ListNode));
    p->next = head;

    struct ListNode *ptr, *ptr1;
    ptr1 = p
    ptr = head
    for(int i=1; i<left; i++){
        ptr1 = ptr;
        ptr = ptr->next;
    }
    struct ListNode *first = ptr;
    struct ListNode *last = ptr1;
    struct ListNode *next = NULL;

    for(int i=left; i<=right; i++){
        struct ListNode *temp = ptr->next;
        ptr->next = next;
        next = ptr;
        ptr = temp;
    }
    last->next = next;

    first->next->ptr;

    return p->next;
}
```

1/2/24

Leetcode 3
//split Linked List in Parts.

```
struct ListNode** splitListToParts(struct ListNode* head,
                                   int k, int* returnSize){
{
    struct ListNode *ptr;
    ptr = head;
    int n=0;
    while (ptr != NULL){
        n++;
        ptr = ptr->next;
    }
    int parts = n/k;
    int extra = n%k;
    struct ListNode** a = (struct ListNode**) malloc
        (k * sizeof (struct ListNode*));
    ptr = head;
    struct ListNode *ptr1 = NULL;
    for (int i=0; i<k; i++){
        a[i] = ptr;
        int size = parts + (extra>0 ? 1 : 0);
        for (int j=0; j<size; j++){
            ptr1 = ptr;
            ptr = ptr->next;
        }
        if (ptr1 != NULL){
            ptr1->next = NULL;
        }
        extra--;
    }
    *returnSize = k;
    return a;
```