18/11/24 Week-5

1. //deletion

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

struct node
{
    int data;
    struct node * next;
}*head = NULL;

void delbeg (struct node * a)
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("List is empty");
    }
    else
    {
        ptr = head;
        head = ptr -> next;
        free (ptr);
        printf("node deleted from beginning");
    }
}

void delend (struct node * a)
{
    struct node *ptr, *ptr1;
    if (head == NULL)
    {
        printf ("List is empty");
    }
```

```
else if(head->next==NULL).
{
    free(head);
    head=NULL;
    printf("only one node is present and its deleted");
}
else
{
    ptr=head;
    while(ptr->next!=NULL)
    {
        ptr1=ptr;
        ptr=ptr->next;
    }
    ptr1->next=NULL;
    free(ptr);
    printf("element deleted at the end");
}
}
void delpos(struct node *a, int pos)
{
    if(head==NULL)
    {
        printf("List is empty");
    }
    else {
        int loc=pos;
        struct node *ptr,*ptr1;
        ptr=head;
        if(loc==1)
        {
            head=ptr->next;
            free(ptr);
```

```
        printf("Deleted at position %d\n", loc);
        return;
    }
    for(int i=0; i<loc-1; i++)
    {
        ptr1 = ptr2;
        ptr2 = ptr2->next;
        if(ptr2==NULL)
        {
            printf("there are less than %d elements", loc);
            return;
        }
    }
    ptr1->next = ptr2->next;
    free(ptr2);
    printf("deleted at %d", loc);
    }
}
void display(struct node *s)
{
    while(s!=NULL)
    {
        printf("%d\t", s->data);
        s=s->next;
    }
}
void create(int a[], int n)
{
    struct node *last, *t;
    head = (struct node*) malloc(sizeof(struct node));
    head->data = a[0];
    head->next = NULL;
```

```
    last=head;
    for(int i=1; i<n; i++)
    {
        t=(struct node *) malloc(sizeof(struct no
        t->data=a[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}
void main()
{
    int a[10], n;
    printf("enter n:");
    scanf("%d",&n);
    printf("enter the values");
    for(int i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
    }
    create(a,n);
    int c,l;
    while(1)
    {
        printf("\n menu\n 1. delete from beg 2. del from end
            3. del at specfic pos 4. display 5. exit");
        printf("enter the choice:");
        scanf("%d",&c);
        switch(c)
        {
            case 1: delbeg(head);
                    break;
```

```
        case 2: delend (head);
                break;
        case 3: printf("enter the loc");
                scanf("%d", &l);
                delpos (head, l);
                break;
        case 4: display (head);
                break;
        case 5: exit(0);
        default: printf("invalid input");
                 break;
    }
  }
}
```

Output

enter n: 5

enter the values: 1 2 3 4 5

menu

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

node deleted from beginning menu

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

2 3 4 5 menu

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

element deleted at the end menu.

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

2 3 4 menu

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

18/11/24

enter the loc 2
deleted at 2 menu
1. delete from beg 2. del from end 3. del at specific
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);
}
```