18/01/24 Week-5

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
 int data;
 struct Node * next;
};
struct node * head=NULL;
void insert();
void begin();
void end();
void anywhos();
void display();
int main()
{
 int choice;
 while(1)
 {
  printf("\n1. Insert elements\n");
  printf("2. Delete at the beginning\n");
  printf("3. Delete at the end\n");
  printf("4. Delete at any position\n");
  printf("5. Display\n");
  printf("6. Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: insert();
           break;
   case 2: begin();
           break;
   case 3: end();
           break;
```

```
case 4: atanyPos();
        break;
case 5: display();
        break;
case 6: exit(0);
default:
    printf("Invalid choice\n");
}
}
return 0;
}
void insert()
{
  struct Node *ptr, *temp;
  int data;
  ptr = (struct Node *)malloc(sizeof(struct Node));
  if (ptr == NULL)
  {
    printf("memory allocation failed\n");
    exit(1);
  }
  printf("Enter data to insert: ");
  scanf("%d", &data);
  ptr->data = data;
  ptr->next = NULL;
  if (head == NULL)
  {
    head = ptr;
    printf("Node inserted successfully\n");
  }
else
  {
    temp = head;
    while (temp->next != NULL)
```

```
}
    temp = temp -> next;
}
temp -> next = ptr;
printf("Node inserted successfully");
}
}
void begin()
{
  struct Node *ptr;
  if (head == NULL)
  {
    printf("List is empty");
  }
  else
  {
    ptr = head;
    head = ptr -> next;
    free(ptr);
    printf("Node deleted at the beginning\n");
  }
}
void end()
{
  struct Node *ptr, *prev;
  if (head == NULL)
  {
    printf("List is empty\n");
  }
  else
  {
    ptr = head;
    while (ptr -> next != NULL)
    {
      prev = ptr;
      ptr = ptr -> next;
    }
```

```
if(ptr==head)
{
  head=NULL;
}
else
{
  prev->next=NULL;
}
free(ptr);
printf("Node deleted from the end\n");
}
}
void arrayfun()
{
  struct Node *ptr, *prev;
  int loc, i=1;
  printf("Enter the position: ");
  scanf("%d", &loc);
  if(head==NULL)
  {
    printf("list is empty\n");
    return;
  }
ptr=head;
if(loc==1)
{
  head=ptr->next;
  free(ptr);
  printf("Node deleted from position %d\n", loc);
  return;
}
```

```
while(ptr != NULL && i<loc)
{
  prev = ptr;
  ptr = ptr -> next;
  i++;
}
if (ptr == NULL)
{
  printf("Position %d is out of bounds\n", loc);
}
else
{
  prev -> next = ptr -> next;
  free(ptr);
  printf("Node deleted from position %d\n", loc);
}
}
void display()
{
  struct Node *ptr;
  if(head == NULL)
  {
    printf("list is empty");
  }
  else
  {
    ptr = head;
    while(ptr != NULL)
    {
      printf("%d", ptr -> data);
      ptr = ptr -> next;
    }
  printf("\n");
  }
}
```

Output

Menu.
1. Insert elements
2. Delete at the beginning
3. Delete at the end
4. Delete at any position
5. Display
6. Exit
Enter your choice: 1
Enter data to insert: 12

Enter your choice: 1
Enter data to insert: 13

Enter your choice: 1
Enter data to insert: 14

Enter your choice: 1
Enter data to insert: 15

Enter your choice: 5
12 13 14 15

Enter your choice: 1
Node deleted from beginning

Enter your choice: 5
13 14 15

Enter your choice: 3
Node deleted at the end

Enter your choice: 5
13 14

18/11/24

leetcode

```
typedef structure{
int size;
int top;
int *s;
int *minstack;
}ministack;

minstack *ministackCreate(){
 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]=value;
  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 in ", 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 minStack(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);
}
```