> lab Program 5:

Singly linked list delete and display displayed implementation:

```
# include <stdio.h>
# include <stdlib.h>
 Struct node {
 int data;
 struct node *next;
 };
 Void display ();
 Void insert_begin();
 Void insert_end();
 Void insert_pos();
 Void begin_delete();
 Struct node *head = NULL;
 Void display()
{
   Printf ("elements are : \n");
   Struct node *Ptr;
  If (head == NULL)
  {
     Printf ("list is empty");
     return;
  }
Else {
     Ptr = head;
     While (Ptr != NULL)
```

```
    {
      Printf ("%d\n", Ptr->data);
      Ptr = Ptr -> next;
    }
  }
}
void insert_begin()
{
  Struct node *temp;
  temp = (struct node*) malloc (sizeof(Structnode));
  printf ("enter the value to be inserted\n");
  Scanf ("%d", &temp->data);
  temp -> next = NULL;
  if (head == NULL)
      head = temp);
  Else {
      temp ->next = head;
      head = temp;
    }
  }
void insert_end()
{
  Struct node *temp, *Ptr;
  temp = (struct node*) malloc (sizeof(Structnode));
  printf ("enter the value to be inserted\n");
  temp -> next = NULL;
  If (head == NULL)
  {
    head = temp;
  }
  Else
    {
```

```
        ptr = head;
        while (ptr -> next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> next = temp;
    }
}
void insert_pos()
{
    int pos, i;
    struct node *temp, *ptr;
    printf("Enter the position");
    scanf("%d", &pos);
    temp = (struct node*) malloc (sizeof(struct node));
    printf("enter the value to be inserted\n");
    scanf("%d", &temp -> data);
    temp -> next = NULL;
    if (pos == 0)
    {
        temp -> next = head;
        head = temp;
    }
    else
    {
        for (i = 0, ptr = head; i < pos - 1; i++)
        {
            ptr = ptr -> next;
        }
        temp -> next = ptr -> next;
        ptr -> next = temp;
    }
}
```

```
void begin_delete()
{
  Struct node *Ptr;
  if (head == NULL)
  { Printf ("\n list is empty \n");
  }
  else
  {
    Ptr = head;
    head = Ptr -> next;
    free (Ptr);
    Printf ("\n Node deleted from the
            begining ... \n");
  }
}
void last_delete()
{
  Struct node *ptr, *Ptr1;
  if (head == NULL)
  {
    Printf ("\n list is empty");
  }
  else if (head -> next == NULL)
  {
    head = NULL;
    free (head);
    Printf ("\n Only node of the list
            deleted... \n");
  }
Else
  {
    Ptr = head;
```

```
While (Ptr -> next != NULL)
 {
   Ptr1 = Ptr;
    Ptr = Ptr -> next;
 }
Ptr1 -> next = NULL;
free (Ptr);
Printf ("\n Deleted Node from the last...\n");
 }
}
void random_delete()
{
 Struct node *Ptr, *Ptr1;
 int loc, i;
 Printf ("\n Enter the location of the node
       after which you want to perform
         deletion\n");
 Scanf ("%d", &loc);
 Ptr= head;
 for (i=0; i< loc; i++)
 {
   Ptr1 =Ptr;
   Ptr = Ptr->next;
   if (Ptr == NULL)
    {
      Printf ("\n can't delete");
      return;
    }
 }
 Ptr1-> next = Ptr -> next;
  free (Ptr);
  Printf ("\n Deleted node %d", loc+1);
  }
```

```
void main ()
{
  int choice;
  while(1)
  {
   Printf ("\n1. to insert at beginning \n"
   "2. to insert at End \n"
   "3. to insrt at Position \n"
   "4. to display \n"
   "5: delete from beginningin"
   "6. delete from End \n"
   "7. random delete \n"
   "8. Exit \n");
Printf ("Enter your choice: \n");
  Scanf("%d", &choice);
  Switch (choice)
  {
   case 1:
       insert_begin();
       break;
   case2:
       insert_End();
       break;
   case 3:
       insert_Pos();
   case 4:
       display();
       break;
   case 5:
       begin_delete();
       break;
```

```
        case 6;
            last_delete();
            break;
        case 7;
            random_delete();
            break;
        Case 8;
            Exit(0);
            break;
        default;
            Printf("invalid choice \n");
            break;
        }
    }
}
```

Output:

1. to insert at beginning
2. to insert at End
3. to insert at Position
4. to display
5. delete from beginning
6. delete from end
7. delete random delete
8. Exit

Enter your choice: 1.
Enter the value to be inserted : 10

Enter your choice: 2
Enter the value to be inserted : 20

Enter your choice : 3
Enter the position : 1
Enter the value to be inserted : 15

Enter your choice : 4
Elements are : 10
15
20

Enter your choice : 5
Node deleted from the beginning ...

Enter your choice : 7
Enter the location of node after which you want to Perform deletion : 1
Deleted node 2

Enter your choice : 6
Only node of the list is deleted ...

Enter your choice : 4
list is empty.

Enter your choice : 8

Exit

18/1/24

111+k4

leetcode : -1

week-3

```
#include <stdlib.h>

typedef Struct{
  int * Stack;
  int * minStack;
  int top
  } MinStack;

MinStack * minStackCreate (){
  MinStack * Stack = (MinStack*)malloc
    (sizeof(MinStack));
  Stack -> stack = (int*) malloc (sizeof(int)*
          10000);
Stack -> minStack = (int* ) malloc (sizeof(int) *10000)

  Stack -> top = -1;
  return stack;
  }
Void minStackpush (MinStack * obj, int val) {
  obj -> top++;
  obj -> stack [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) {
    obj->top--;
}

int minStackTop (MinStack* obj) {
    return obj->stack[obj->top];
}

int minStackGetMin (MinStack* obj) {
    return obj->minStack[obj->top];
}

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

Output: ["MinStack", "Push", "Push", "Push", "getMin", "Pop", "top", "getMin"]

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

Output: [null, null, null, null, -3, null, 0, -2]

NLP
18/11/24

```
#include<stdlib.h>
struct node{
int data;
struct node*next;
};
void display();
void insert_begin();
void insert_end();
void insert_pos();
void begin_delete();
struct node *head=NULL;
void display()
{

    printf("elements are :\n");
    struct node *ptr;
    if(head==NULL)
    {

        printf("list is empty");
        return;


    }
    else{
        ptr=head;
        while(ptr !=NULL)
        {

            printf("%d\n", ptr->data);
            ptr=ptr->next;
        }
    }
}
void insert_begin()
{

    struct node*temp;
    temp =(struct node*)malloc(sizeof(struct node));
    printf("enter the value to be inserted\n");
    scanf("%d",&temp->data);
    temp->next=NULL;
    if(head==NULL)
        head=temp;
    else{
        temp->next=head;
        head=temp;
    }
```

```
    temp->next=NULL;
    if(head==NULL)
        head=temp;
    else{
        temp->next=head;
        head=temp;
    }
}
void insert_end()
{

    struct node *temp,*ptr;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("enter the value to be inserted \n");
    scanf("%d",&temp->data);
    temp->next=NULL;
    if(head==NULL)
    {

        head=temp;
    }
    else
    {

        ptr=head;
        while(ptr->next != NULL)
        {

            ptr=ptr->next;
        }
        ptr->next=temp;
    }
}
void insert_pos()
{

    int pos,i;
    struct node*temp,*ptr;
    printf("enter the position");
    scanf("%d",&pos);
    temp=(struct node*)malloc(sizeof(struct node));
    printf("enter the value to be inserted\n");
    scanf("%d",&temp->data);
    temp->next=NULL;
    if(pos==0)
    {
```

```
    temp=(struct node*)malloc(sizeof(struct node));
    printf("enter the value to be inserted\n");
    scanf("%d",&temp->data);
    temp->next=NULL;
    if(pos==0)
    {

        temp->next=head;
        head=temp;

    }
    else
    {

        for(i=0, ptr=head; i<pos-1;i++)
        {

            ptr=ptr->next;
        }
        temp->next=ptr->next;
        ptr->next=temp;
    }


oid begin_delete()

    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }


oid last_delete()

    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
```

```
oid last_delete()

    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }

oid random_delete()

    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;

        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
```

```
        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted node %d ",loc+1);


oid main()


    int choice;
    while(1)
    {

        printf("\n 1.to insert at the beginning\n"
               " 2.to insert at the end\n "
               "3.to insert at the position\n "
               "4.to display\n "
               "5.delete from beginning\n"
               "6.delete from end\n"
               "7.random delete\n"
               "8.exit\n");
        printf("enter you choice:\n");
        scanf("%d",&choice);
        switch(choice)
        {

        case 1:
            insert_begin();
            break;
        case 2:
            insert_end();
            break;
        case 3:
            insert_pos();
            break;
        case 4:
            display();
            break;
        case 5:
            begin_delete();
            break;
```

```
    printf("\n 1.to insert at the beginning\n"
           " 2.to insert at the end\n "
           "3.to insert at the position\n "
           "4.to display\n "
           "5.delete from beginning\n"
           "6.delete from end\n"
           "7.random delete\n"
           "8.exit\n");
    printf("enter you choice:\n");
    scanf("%d",&choice);
    switch(choice)
    {

    case 1:
        insert_begin();
        break;
    case 2:
        insert_end();
        break;
    case 3:
        insert_pos();
        break;
    case 4:
        display();
        break;
    case 5:
        begin_delete();
        break;
    case 6:
        last_delete();
        break;
    case 7:
        random_delete();
        break;
    case 8:
        exit(0);
        break;
    default:
        printf("invalid choice\n");
        break;
    }
}
```

```
 1.to insert at the beginning
 2.to insert at the end
 3.to insert at the position
 4.to display
 5.delete from beginning
6.delete from end
7.random delete
8.exit
enter you choice:
3
enter the position1
enter the value to be inserted
30

 1.to insert at the beginning
 2.to insert at the end
 3.to insert at the position
 4.to display
 5.delete from beginning
6.delete from end
7.random delete
8.exit
enter you choice:
4
elements are :
10
30
20

 1.to insert at the beginning
 2.to insert at the end
 3.to insert at the position
 4.to display
 5.delete from beginning
6.delete from end
7.random delete
8.exit
enter you choice:
5

Node deleted from the begining ...

 1.to insert at the beginning
 2.to insert at the end
 3.to insert at the position
 4.to display
 5.delete from beginning
6.delete from end
7.random delete
8.exit
enter you choice:
7

 Enter the location of the node after which you want to perform deletion
1

Deleted node 2
 1.to insert at the beginning
 2.to insert at the end
 3.to insert at the position
 4.to display
 5.delete from beginning
6.delete from end
```

```
lements are :
0
0
0

1.to insert at the beginning
2.to insert at the end
3.to insert at the position
4.to display
5.delete from beginning
.delete from end
.random delete
.exit
nter you choice:


ode deleted from the begining ...

1.to insert at the beginning
2.to insert at the end
3.to insert at the position
4.to display
5.delete from beginning
.delete from end
.random delete
.exit
nter you choice:


Enter the location of the node after which you want to perform deletion


eleted node 2
1.to insert at the beginning
2.to insert at the end
3.to insert at the position
4.to display
5.delete from beginning
.delete from end
.random delete
.exit
nter you choice:

lements are :
0

1.to insert at the beginning
2.to insert at the end
3.to insert at the position
4.to display
5.delete from beginning
.delete from end
.random delete
.exit
nter you choice:
```

11/1/24

leetcode -1

Week-3

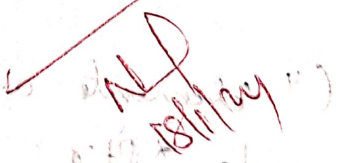
```
#include <stdlib.h>

typedef Struct{
  int * Stack;
  int * min Stack;
  int top
  } MinStack;

MinStack * minStack Create (){
 MinStack * Stack = (minStack*) malloc
    (sizeof (MinStack));
Stack -> Stack = (int*) malloc (size of (int)*
            10000);
Stack -> minStack = (int* ) malloc (sizeof (int) *10000);

 Stack -> top = -1;
 return stack;
 }
Void minStackpush (MinStack * Obj, int val) {
 obj -> top++;
 obj -> Stack [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) {
    obj->top--;
}
int minStackTop (MinStack* obj) {
    return obj->stack[obj->top];
}
int minStackGetMin (MinStack* obj) {
    return obj->minStack[obj->top];
}
void minStackFree (MinStack* obj) {
    free (obj->stack);
    free (obj->minStack);
    free (obj);
}
```

Output: ["MinStack", "Push", "Push", "Push", "getMin", "Pop", "top", "getMin"]

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

Output: [null, null, null, null, -3, null, 0, -2]



```c
#include <stdlib.h>


typedef struct {
int *stack;
int *minStack;
int top
} MinStack;


MinStack* minStackCreate() {
MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
stack->stack = (int*)malloc(sizeof(int) * 10000);
stack->minStack = (int*)malloc(sizeof(int) * 10000);
stack->top = -1;
return stack;
}

void minStackPush(MinStack* obj, int val) {
obj->top++;
obj->stack[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) {
obj->top--;
}
```

```c
obj->top++;
obj->stack[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) {
obj->top--;
}

int minStackTop(MinStack* obj) {
return obj->stack[obj->top];
}

int minStackGetMin(MinStack* obj) {
return obj->minStack[obj->top];
}

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

- Case 1

Input

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

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

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

```
[null,null,null,null,-3,null,0,-2]
```