Lab 7 — Singly linked list :-

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

struct node
 {
  int info;
  struct node *link;
 };
typedef struct node *NODE;

NODE getnode()
{
 NODE x;
 x = (NODE) malloc(sizeof(struct node));
 if (x == NULL)
  {
   printf("mem full\n");
   exit(0);
  }
 return x;
 }
void freenode(NODE x)
 {
 free(x);
   }
NODE insert_front(NODE first, int item)
{
```

```
NODE temp;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
  return temp;
temp -> link = first;
first = temp;
return first;
}
NODE delete_front (NODE first)
{
NODE temp;
if (first == NULL)
{
printf ("list is empty cannot delete\n");
return first;
}
temp = first;
temp = temp -> link;
printf ("item deleted at front-end is = %d\n",
                                    first -> info);
free (first);
return temp;
}
NODE insert_rear (NODE first, int item)
{
NODE temp, cur;
```

```
temp = getnode ();
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
    return temp;
cur = first;
while ( cur -> link != NULL)
    cur = cur -> link;
cur -> link = temp;
return first;
}
NODE delete_rear (NODE first)
{
NODE cur, prev;
if ( first == NULL)
{
printf ("list empty cannot delete \n");
return first;
}
if ( first -> link == NULL)
{
printf ("item deleted is %d\n", first -> info);
free (first);
return NULL;
}
prev = NULL;
cur = first;
while ( cur -> link != NULL)
{
```

```
    prev = cur;
    cur = cur -> link;
  }
  printf("item deleted at rear end is %d", cur->info);
  free(cur);
  prev -> link = NULL;
  return first;
}
NODE insert_pos(int item, int pos, NODE first)
{
  NODE temp;
  NODE prev, cur;
  int count;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL && pos == 1)
    return temp;
  if (first == NULL)
  {
    printf("invalid pos\n");
    return first;
  }
  if (pos == 1)
  {
    temp -> link = first;
    return temp;
  }
  count = 1;
```

```
prev = NULL;
cur = first;
while ( cur != NULL && count != pos)
{
prev = cur;
cur = cur -> link;
count++;
}
if ( count == pos)
{
prev -> link = temp;
temp -> link = cur;
return first;
}
printf ("IP\n");
return first;
}
NODE delete_pos (int pos, NODE first)
{
if (first == NULL)
{
printf ("list empty\n");
return first;
}
NODE temp = first;
if (pos == 1)
{
```

```
        first = temp -> link;
        free (temp);
        return first;
    }
NODE prev;
    for (int i=1; temp != NULL && i<pos; i++);
        prev = temp;
        temp = temp -> link;
    }
if (temp == NULL || temp -> link == NULL){
        printf ("Invalid position\n");
        return NULL;
    }
    prev -> link = temp -> link;
printf ("Element deleted %d\n", temp -> info);
    free (temp);
    return first;
}
void display (NODE first)
{
NODE temp;
if (first == NULL)
    printf ("List empty\n");
for (temp = first; temp != NULL; temp = temp->
    printf ("%d\n", temp -> info);
    }
}
```

```
NODE concat (NODE first, NODE second)
{
 NODE cur;
  if (first == NULL)
     return second;
  if (second == NULL)
     return first;
  cur = first;
  while (cur -> link != NULL)
     cur = cur -> link;
  cur -> link = second;
    return first;
}
NODE reverse (NODE first)
{
 NODE cur, temp;
  Cur = NULL;
while (first != NULL)
  {
    temp = first;
    first = first -> link;
    temp -> link = Cur;
      cur = temp;
  }
 return Cur;
 }
```

```
NODE order_list (NODE first)
{
    int swapped, i;
    NODE ptr1, lptr = NULL;

    if (first == NULL)
        return first;

do
{
    swapped = 0;
    ptr1 = first;
    while (ptr1 -> link != lptr)
    {
        if (ptr -> info > ptr1 -> link -> info)
        {
            int temp = ptr1 -> info;
            ptr1 -> info = ptr1 -> link -> info;
            ptr1 -> link -> info = temp;
            swapped = 1;
        }
        ptr1 = ptr1 -> link;
    }
    lptr = ptr1;
}
while (swapped);
return first;
}
```

```
void main ()
{
int item, choice, pos, i, n;
NODE a, b;
NODE first;

for (;;)
{
printf ("1. insert front \n
        2. delete front \n
        3. insert rear \n
        4. delete rear \n
        5. insert pos \n
        6. delete pos \n
        7. concat \n
        8. reverse \n
        9. order list \n
        10. display ");
printf ("Enter choice \n");
 Scanf ("%d", &choice);
switch (choice)
{
case 1: printf ("Enter item ");
        Scanf ("%d", &item);
        first = insert_front (first, item);
        break;
```

```
case 2: first = delete_front (first);
        break;
case 3: printf ("enter the item at rear-end\n");
        scanf ("%d", &item);
        first = insert_rear (first, item);
        break;
case 4: first = delete_rear (first);
        break;
case 5:
    printf ("Enter item\n");
    scanf ("%d", &item);
    printf ("Enter the position\n");
        scanf ("%d", &pos);
        first = insert_pos (item, pos, first);
        break;
case 6:
    printf ("Enter position of deletion\n");
    scanf ("%d", &pos);
        first = delete_pos (pos, first);
        break;
case 7:
    printf ("enter no. of nodes in 1\n");
        scanf ("%d", &n);
        a = NULL;
        for (i = 0; i < n; i++)
        {
            printf ("Enter item\n");
            scanf ("%d", &item);
```

```
        a = insert_rear(a, item);
        }
    printf("enter no. of nodes in 2\n");
      scanf("%d", &n);
    b = NULL;
for(i=0; i<n; i++)
  {
    printf("enter item\n");
      scanf("%d", &item);
      b = insert_rear(b, item);
  }
  a = concat(a, b);
  display(a);
  break;

case 8:
  first = reverse(first);
    break;
case 9:
      first = order_list(first);
      break;
 case 10: display(first);
        break;
 default: exit(0);
        break;
    }
  }
}
```