Q1:linked list all basic operations menu driven program..
ANS:-
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
typedef struct node
{
    int data;
    struct node *next;
}NODE;

NODE *create_list(NODE *list)
{
    NODE *temp, *new_node;
    int n, i;
    printf("\n Enter how many nodes you want:");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        new_node = (NODE *)malloc(sizeof(struct node));
        new_node->next = NULL;
        printf("\n Enter [%d] list element:", i);
        scanf("%d", &new_node->data);
        if (list == NULL)
        {
            list = temp = new_node;
        }
        else
        {
            temp->next = new_node;
            temp = new_node;
        }
    }
    return list;
}
void display(NODE *list)
{
    while (list != NULL)
    {
        printf("\t%d", list->data);
        list = list->next;
    }
}
NODE *insert(NODE *list, int p, int n)
{
    NODE *temp, *new_node;
    int i;
    new_node = (NODE *)malloc(sizeof(struct node));
    new_node->data = n;
    new_node->next = NULL;
    if (p == 1)
    {
        new_node->next = list;
        list = new_node;
    }
    else
    {
        for (i = 1, temp = list; i < p - 1 && temp != NULL; i++, temp =
temp->next);

        if (temp == NULL)
        {
            printf("\n position out of range!");
            return list;
        }
        else
        {
          new_node->next = temp->next;
          temp->next = new_node;

      }
    }
    return list;
}
NODE *deleteByValue(NODE *list, int n)
{
    NODE *temp = list, *temp1;
    if (list->data == n)
    {
      list = temp->next;
      free(temp);
      return list;
    }

    while (temp->next != NULL)
    {
      if (temp->next->data == n)
      {
          temp1 = temp->next;
          temp->next = temp1->next;
          free(temp1);
          return list;
      }
      temp = temp->next;
    }
    // return list;//if all similar element have to remove from list!
    if (temp->next == NULL)
    {
      printf("\n The element not found!");
    }
    return list;
}
NODE *deleteByPos(NODE *list, int p)
{
    NODE *temp = list, *temp1;
    int i;
    if (p == 1)
    {
        list = temp->next;
        free(temp);
        return list;
    }

    for (i = 1, temp = list; temp->next != NULL && i < p - 1; i++, temp =
temp->next);
```

```
    if (temp == NULL)
    {
        printf("\n Position out of range!");
        return list;
    }
    temp1 = temp->next;
    temp->next = temp1->next;
    free(temp1);
    return list;
}
NODE *search(NODE *list, int n)
{
    NODE *temp = list;
    while (temp != NULL)
    {
      if (temp->data == n)
      {
          return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
NODE *reverse(NODE *list)
{
    NODE *t1, *t2, *t3;
    t1 = list;
    t2 = t1->next;
    t3 = t2->next;
    t1->next = NULL;
    while (t3 != NULL)
    {
        t2->next = t1;
        t1 = t2;
        t2 = t3;
        t3 = t3->next;
    }
    t2->next = t1;
    return t2;
}
NODE *concat(NODE *list, NODE *list1)
{
    NODE *temp;
    if (list == NULL)
        return list1;
    if (list1 == NULL)
        return list;
    for (temp = list; temp->next != NULL; temp = temp->next);
    temp->next = list1;
    return list;
}
NODE*insertAtBeginning(NODE*list,int n)
{
    NODE*new_node;
    new_node=(NODE*)malloc(sizeof(NODE));
    new_node->data=n;
    new_node->next=list;
    list=new_node;
    return list;

}
NODE*intersection(NODE*list1,NODE*list2,NODE*list)
{
    NODE*temp=list1;
    NODE*temp1=list2;

    for(temp=list1;temp!=NULL;temp=temp->next)
    {
      for(temp1=list2;temp1!=NULL;temp1=temp1->next)
      {
          if (temp->data==temp1->data)
          {
            list=insertAtBeginning(list,temp->data);
            break;
          }

      }
    }
    return list;

}
int main()
{
    NODE *list = NULL, *list1 = NULL,*list2=NULL,*temp;
    int choice, pos, num;
    do
    {
      printf("\n------------------------------------");
      printf("\n 1.create \n 2.display \n3.insert \n4.delete by value \n
5.delete by position \n6.Search \n7.Reverse
\n8.Concatenation\n9.Intersection\n10.Exit");
      printf("\n------------------------------------");
      printf("\n Enter your choice:");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
          list = create_list(list);
          break;
      case 2:
          display(list);
          break;
      case 3:
          printf("\n Enter position and data for inserting in list:");
          scanf("%d%d", &pos, &num);
          list = insert(list, pos, num);
          break;
      case 4:
          printf("\n Enter value for deleting into list:");
          scanf("%d", &num);
          list = deleteByValue(list, num);
          break;
      case 5:
          printf("\n Enter position for delete the element:");
          scanf("%d", &pos);
```

```
            list = deleteByPos(list, pos);
            break;
        case 6:
            printf("\n Enter element to search:");
            scanf("%d", &num);
            temp = search(list, num);
              if (temp == NULL)
              {
                  printf("\n The element not found!");
              }
              else
              {
                  printf("\n Element founded at %u node address!", temp);
              }
              break;
          case 7:
              list = reverse(list);
              printf("\n The Reverse list is:\n");
              display(list);
              break;
          case 8:
              list1 = create_list(list1);
              list = concat(list, list1);
              printf("\n After concatenate list is:\n");
              display(list);
              break;
          case 9:
              list1 = create_list(list1);
              list2=intersection(list,list1,list2);
              printf("\n The intersection of two list are:\n");
              display(list2);
              break;
          default:
              printf("\n Wrong input!");
              break;
          }
      } while (choice != 10);
      getch();
      return 0;
}
```