```
/*P11.14 Program of linked list*/
#include<stdio.h>
#include<stdlib.h>
struct node
{
     int info;
     struct node *link;
};
struct node *create_list(struct node *start);
void display(struct node *start);
void count(struct node *start);
void search(struct node *start,int data);
struct node *addatbeg(struct node *start,int data);
struct node *addatend(struct node *start,int data);
struct node *addafter(struct node *start,int data,int item);
struct node *addbefore(struct node *start,int data,int item);
struct node *addatpos(struct node *start,int data,int pos);
struct node *del(struct node *start,int data);
struct node *reverse(struct node *start);
int main(void)
{
     struct node *start=NULL;
     int choice,data,item,pos;
     while(1)
     {
          printf("1.Create List\n");
          printf("2.Display\n");
          printf("3.Count\n");
          printf("4.Search\n");
          printf("5.Add to empty list / Add at beginning\n");
          printf("6.Add at end\n");
          printf("7.Add after node\n");
          printf("8.Add before node\n");
          printf("9.Add at position\n");
          printf("10.Delete\n");
          printf("11.Reverse\n");
          printf("12.Quit\n\n");
          printf("Enter your choice : ");
          scanf("%d",&choice);

          switch(choice)
          {
           case 1:
                start=create_list(start);
                break;
           case 2:
                display(start);
                break;
           case 3:
                count(start);
                break;
           case 4:
                printf("Enter the element to be searched : ");
                scanf("%d",&data);
                search(start,data);
                break;
           case 5:
                printf("Enter the element to be inserted : ");
```

```
                scanf("%d",&data);
                start=addatbeg(start,data);
                break;
           case 6:
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                start=addatend(start,data);
                break;
           case 7:
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                printf("Enter the element after which to insert : ");
                scanf("%d",&item);
                start=addafter(start,data,item);
                break;
           case 8:
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                printf("Enter the element before which to insert: ");
                scanf("%d",&item);
                start=addbefore(start,data,item);
                break;
           case 9:
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                printf("Enter the position at which to insert : ");
                scanf("%d",&pos);
                start=addatpos(start,data,pos);
                break;
           case 10:
                printf("Enter the element to be deleted : ");
                scanf("%d",&data);
                start=del(start, data);
                break;
           case 11:
                start=reverse(start);
                break;
           case 12:
                 exit(1);
           default:
                 printf("Wrong choice\n");
          }/*End of switch*/
     }/*End of while*/
     return 0;
}/*End of main()*/

 .....display()  or Traverse function:.....
void display(struct node *start)
{
     struct node *p;
     if(start==NULL)
     {
          printf("List is empty\n");
          return;
     }
     p=start;
     printf("List is :\n");
```

```
     while(p != NULL)
     {
          printf("%d ",p->info);
          p=p->link;
     }
     printf("\n\n");
}
/*End of display() */
/.....................................................count()
function,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
void count(struct node *start)
{
     struct node *p;
     int cnt=0;
     p=start;
     while(p!=NULL)
     {
          p = p->link;
          cnt++;
     }
     printf("Number of elements are %d\n",cnt);
}
/*End of count() */
.....................................search()
function.....................................................................

void search(struct node *start,int item)
{
     struct node *p=start;
     int pos=1;
     while(p!=NULL)
     {
          if(p->info == item)
          {
               printf("Item %d found at position %d\n",item,pos);
               return;
          }
          p=p->link;
          pos++;
     }
     printf("Item %d not found in list\n",item);
}/*End of search()*/

.....................................................addatbeg()
function.....................................................

struct node *addatbeg(struct node *start,int data)
{
     struct node *tmp;
     tmp=(struct node *)malloc(sizeof(struct node));
     tmp->info=data;
     tmp->link=start;
     start=tmp;
     return start;
}/*End of addatbeg()*/
.....................................addatend().....................
.....................................
struct node *addatend(struct node *start,int data)
```

```
{
      struct node *p,*tmp;
      tmp=(struct node *)malloc(sizeof(struct node));
      tmp->info=data;
      p=start;
      while(p->link!=NULL)
            p=p->link;
      p->link=tmp;
      tmp->link=NULL;
      return start;
}/*End of addatend()*/
```

.....addafter().....

```
struct node *addafter(struct node *start,int data,int item)
{
      struct node *tmp,*p;
      p=start;
      while(p!=NULL)
      {
            if(p->info == item)
            {
                  tmp=(struct node *)malloc(sizeof(struct node));
                  tmp->info=data;
                  tmp->link=p->link;
                  p->link=tmp;
                  return start;
            }
            p=p->link;
      }
      printf("%d not present in the list\n",item);
      return start;
}/*End of addafter()*/
```

addbefore(),,,,,

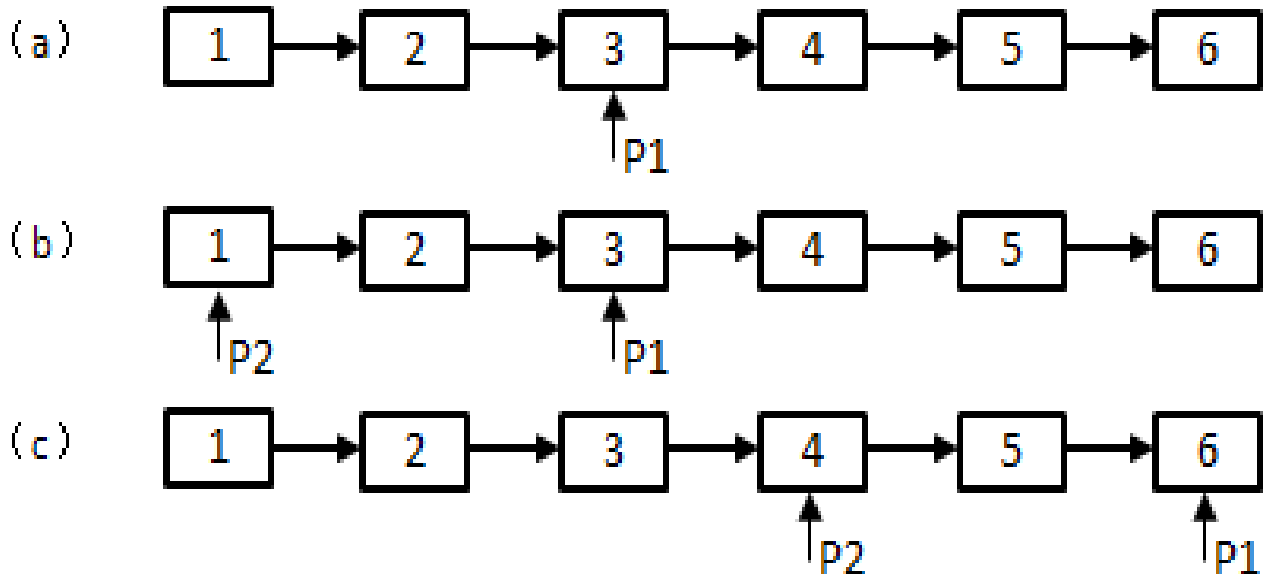
```
struct node *addbefore(struct node *start,int data,int item)
{
      struct node *tmp,*p;
      if(start == NULL )
      {
            printf("List is empty\n");
            return start;
      }
      /*If data to be inserted before first node*/
      if(item==start->info)
      {
            tmp=(struct node *)malloc(sizeof(struct node));
            tmp->info=data;
            tmp->link=start;
            start=tmp;
            return start;
      }
      p = start;
      while(p->link!=NULL)
      {
            if(p->link->info==item)
```

```
          {
               tmp=(struct node *)malloc(sizeof(struct node));
               tmp->info=data;
               tmp->link=p->link;
               p->link=tmp;
               return start;
          }
          p=p->link;
     }
     printf("%d not present in the list\n",item);
     return start;
}/*End of addbefore()*/
.................................................addatpos()*.................
.............................................................

struct node *addatpos(struct node *start,int data,int pos)
{
     struct node *tmp,*p;
     int i;
     p=start;
     for(i=1; i<pos-1 && p!=NULL; i++)
          p=p->link;
     if(p==NULL)
          printf("There are less than %d elements\n",pos);
     else
     {
          tmp=(struct node *)malloc(sizeof(struct node));
          tmp->info=data;
          if(pos==1)
          {
            tmp->link=start;
            start=tmp;
          }
        else
          {
           tmp->link=p->link;
           p->link=tmp;
          }
     }
    return start;
}/*End of addatpos()*/

.....................................
create_list()..............................................................
..........
struct node *create_list(struct node *start)
{
     int i,n,data;
     printf("Enter the number of nodes : ");
     scanf("%d",&n);
     start=NULL;
     if(n==0)
          return start;
     printf("Enter the element to be inserted : ");
     scanf("%d",&data);
     start=addatbeg(start,data);
     for(i=2; i<=n; i++)
     {
```

```
            printf("Enter the element to be inserted : ");
            scanf("%d",&data);
            start=addatend(start,data);
      }
      return start;
}/*End of create_list()*/
.........................................
delete()..................................................................
........

struct node *del(struct node *start,int data)
{
      struct node *tmp,*p;
      if(start==NULL)
      {
            printf("List is empty\n");
            return start;
      }
      if(start->info==data)  /*Deletion of first node*/
      {
            tmp=start;
            start=start->link;
            free(tmp);
            return start;
      }
      p=start;   /*Deletion in between or at the end*/
      while(p->link!=NULL)
      {
            if(p->link->info==data)
            {
                  tmp=p->link;
                  p->link=tmp->link;
                  free(tmp);
                  return start;
            }
            p=p->link;
      }
      printf("Element %d not found\n",data);
      return start;
}/*End of del()*/
.........................................reverse().....................
.................................................
struct node *reverse(struct node *start)
{
      struct node *prev,*ptr,*next;
      prev=NULL;
      ptr=start;
      while(ptr!=NULL)
      {
            next=ptr->link;
            ptr->link=prev;
            prev=ptr;
            ptr=next;
      }
      start=prev;
      return start;
}/*End of reverse()*/
```

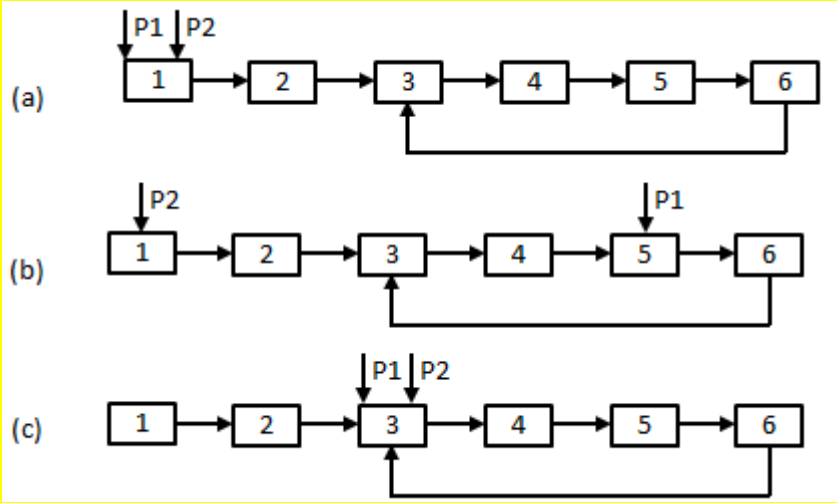
```
.........................n-th Node from End ..........................................................
struct Node
{
  int Value;
   struct Node* Next;
};

ListNode* Find nthfrom end(ListNode* start, unsigned int n)
{
   if(start == NULL || n == 0)
     return NULL;

   ListNode *p = start;
   ListNode *q= NULL;

  for(unsigned int i = 0; i < n;i++)
  {
    if(p->Next != NULL)
    p = p->Next;
     else
    {
    return NULL;
     }
   }

   q = start;
  while(p->Next != NULL)
  {
      p = p->Next;
     q = q->Next;
  }

 return q;
}
\
          or
/* Function to get the nth node from the last of a linked list*/
void printNthFromLast(struct node* head, int n)
{
     int len = 0, i;
    struct node *temp = head;

 // 1) count the number of nodes in Linked List
 while (temp != NULL)
    {
    temp = temp->next;
     len++;
    }

 // check if value of n is not more than length of the linked list
     if (len < n)
     return;
temp = head;
// 2) get the (n-len+1)th node from the begining
```

```
   for (i = 1; i < len-n+1; i++)
   temp = temp->next;

   printf ("%d", temp->data);
   return;
}
```




```
..... find Loop in List .....
int detectloop(struct node *start)
{
struct node *p = start, *q = start;

 while(p && q && q->next )
  {
    p = p->next;
    q= q->next->next;
     if (p == q)
     {
       printf("Found Loop");
       return 1;
    }
   }
  return 0;
}

          or

struct Loop(structNode* start)
{
   if(start == NULL)
  return false;

  struct Node* p = start->Next;
```

```
   if(p== NULL)
      return false;

    struct Node* q = p->Next;
      while(q != NULL && p!= NULL)
     {
        if(q == p)
         return true;

         p = p->Next;

             q = q->Next;
             if(q != NULL)
               q =q->Next;
      }

   return false;
}
```



.....meeting node .....

```
struct Node* MeetingNode(struct Node* start)
{
  if(start == NULL)
     return NULL;

  struct Node* p = start->Next;
  if(p == NULL)
  return NULL;

  structNode* q = p->Next;
   while(q != NULL && p != NULL)
  {
          if(q == p)
            returnq q;
      p = p->Next;
      q = q->Next;
         if(q != NULL)
         q = q->Next;
   }
```

```
  return NULL;
}
```

.....entry node in a loop.....

```
struct Node* EntryNodeOfLoop(struct Node* start)
{
  struct Node* meetingNode = MeetingNode(start);
  if(meetingNode == NULL)
     return NULL;

  // get the number of nodes in loop
    int nodesInLoop = 1;
  struct Node* p = meetingNode;
  while(p->Next != meetingNode)
    {
         p = p->Next;
        ++nodesInLoop;
     }

      // move pNode1
  p = start;
   for(int i = 0; i < nodesInLoop; ++i)
   p = p->Next;

   // move pNode1 and pNode2
  struct Node* q= start;
   while(p != q)
    {
   p = p->Next;
    q = q->Next;
   }

   return p;
}
```

Delete N nodes after M nodes of a linked list

Input:
M = 3, N = 2
Linked List: 1->2->3->4->5->6->7->8->9->10
Output:
Linked List: 1->2->3->6->7->8

```
struct node
{
    int data;
    struct node *next;
};
```

```
// Function to skip M nodes and then delete N nodes of the linked list.

void skipMdeleteN(struct node  *start, int M, int N)
{
    struct node *p = start, *q;
    int count;

    while (p)
    {
        for (count = 1; count<M && p!= NULL; count++)
            p= p->next;

        if (p== NULL)
            return;

              q= p->next;
        for (count = 1; count<=N && q!= NULL; count++)
        {
            struct node *temp = q;
            q = q->next;
            free(temp);
        }
        p->next = q; // Link the previous list with remaining nodes

        // Set current pointer for next iteration
        p = q;
    }
}
```

Swap nodes in a linked list without swapping data

```
struct node
{
  int data;
 struct node *next;
};

/*Function to swap nodes x and y in linked list by changing links
*/
void swapNodes(struct node **head_ref, int x, int y)
{
 // Nothing to do if x and y are same
   if (x == y) return;

 // Search for x (keep track of prevX and CurrX
 struct node *prevX = NULL, *currX = *head_ref;
  while (currX && currX->data != x)
  {
```

```
   prevX = currX;
   currX = currX->next;
  }

   // Search for y (keep track of prevY and CurrY
  struct node *prevY = NULL, *currY = *head_ref;
    while (currY && currY->data != y)
    {
      prevY = currY;
    currY = currY->next;
   }

   // If either x or y is not present, nothing to do
   if (currX == NULL || currY == NULL)
       return;

   // If x is not head of linked list
   if (prevX != NULL)
       prevX->next = currY;
   else // Else make y as new head
       *head_ref = currY;

   // If y is not head of linked list
   if (prevY != NULL)
       prevY->next = currX;
   else // Else make x as new head
       *head_ref = currX;

   // Swap next pointers
   struct node *temp = currY->next;
   currY->next = currX->next;
   currX->next  = temp;
}
```

how to find linked list is circular or not in c

```
int circular(/*struct node *head*/){
    if(head==NULL)
        return 0;
    struct node *fast=head, *slow=head;
    while(fast && fast->next){
        if(fast->next->next==slow)
            return 1;
        fast=fast->next->next;
        slow=slow->next;
    }
    return 0;
}
```

```
/* Function to get the middle of the linked list*/
void printMiddle(struct node *head)
{
    struct node *slow_ptr = head;
    struct node *fast_ptr = head;

    if (head!=NULL)
    {
        while (fast_ptr != NULL && fast_ptr->next != NULL)
        {
            fast_ptr = fast_ptr->next->next;
            slow_ptr = slow_ptr->next;
        }
        printf("The middle element is [%d]\n\n", slow_ptr->data);
    }
}
```