```c
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
struct node
{
    int data;
    struct node *next;
};

struct node *list_creation(int n)
{
    struct node *head, *temp, *new;
    int item;
    head = (struct node *)malloc(sizeof(struct node));
    printf("Enter data in node\n");
    scanf("%d", &item);
    head->data = item;
    head->next = NULL;
    temp = head;
    for (int i = 1; i < n; i++)
    {
        new = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &item);
        new->data = item;
        new->next = NULL;
        temp->next = new;
        temp = temp->next; // to jump on next node
    }

    return head;
}

void list_traversal(struct node *headptr)
{
    struct node *tail;
    if (headptr != NULL)
    {
        tail = headptr;
        printf("\n Data in linked list is\n");
        while (tail != NULL)
        {
            printf("%d\t", tail->data);
            tail = tail->next;
        }
    }
    else
    {
        printf("\nThere is no data\n");
    }
}

struct node *insert_end(struct node *headptr)
{
    int item, size;
    struct node *end, *temp;
    temp = headptr;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    printf("\nhow many elemnent you want to insert at at the end of the linked
list\t");
    scanf("%d", &size);
    printf("\nenter value for new node to add at end\n");
    for (int i = 0; i < size; i++)
    {
        end = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &item);
        end->data = item;
        end->next = NULL;
        temp->next = end;
        temp = temp->next;
    }
    return headptr;
}

struct node *insert_begin(struct node *headptr)
{
    int item, size;
    struct node *temp;
    printf("\nhow many elemnent you want to insert at at the start of the linked
list\t");
    scanf("%d", &size);
    printf("\nenter value for new node to add at start\n");
    for (int i = 0; i < size; i++)
    {
        temp = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &item);
        temp->data = item;
        temp->next = headptr;
        headptr = temp;
    }
    return headptr;
}

struct node *insert_middle(int p, struct node *headptr)
{
    int item, i = 1;
    struct node *temp, *curr;
    if (p == 1 || p == 0)
    {
        return insert_begin(headptr);
    }
    curr = headptr;
    while (i < p - 1)
    {
        curr = curr->next;
        if (curr->next == NULL)
        {
            return insert_end(headptr);
            break;
        }
        i++;
    }

    printf("\nenter value for new node to add at %dth posion\n", p);
    temp = (struct node *)malloc(sizeof(struct node));
    scanf("%d", &item);
    temp->data = item;
    temp->next = curr->next;
    curr->next = temp;

    return headptr;
}

struct node *delete_head(struct node *headptr)
{
    struct node *temp;
    if (headptr != NULL)
    {
        temp = headptr;
        headptr = headptr->next;
        free(temp);
        return headptr;
    }
    else
    {
        printf("\nThere is no data to delete \n");
    }
}

struct node *delete_end(struct node *headptr)
{
    struct node *temp, *curr;
    curr = headptr;
    while (curr->next->next != NULL)
    {

        curr = curr->next;
    }
    temp = curr->next;
    curr->next = NULL;
    free(temp);
    return headptr;
}

struct node *delete_middle(struct node *headptr, int pos)
{
    struct node *temp, *curr;
    int i = 1;
    curr = headptr;
    if (pos == 1 || pos == 0)
    {
        return delete_head(headptr);
    }
    while (i < pos - 1)
    {
        curr = curr->next;
        if (curr->next == NULL)
        {
            return delete_end(headptr);
            break;
        }
        i++;
    }
    temp = curr->next;
    curr->next = curr->next->next;
    free(temp);
    return headptr;
}

struct node *reverse(struct node *prev, struct node *curr)
{
    struct node *next;
    next = curr->next;
    curr->next = prev;
    if (next == NULL)
    {
        return curr;
    }
    else
    {
        reverse(curr, next);
    }
}

struct node *concat(struct node *head1, struct node *head2)
{
    struct node *temp;
    temp = head1;
    while (temp != NULL)
    {
   if (temp->next==NULL){
     temp->next = head2;
     break;
   }
   else{
      temp = temp->next;
   }
    }
    return head1;
}
void main()
{
    int size, choice, pos;
    struct node *N, *M;
    printf("how many elemnent you want in linked list\t");
    scanf("%d", &size);
    N = list_creation(size);
```

```
label:
printf("\nPress enter to continue\n");
getch();
system("cls");
    printf("\n Press 1 for traversal\n Press 2 for insert element at end \n Press 3 for 
insert element at start \n Press 4 for insert element at middle");
    printf("\n Press 5 for Deleting element at head  \n Press 6 Deleting element at end 
\n Press 7 Deleting element at middle \n Press 8 to reverse the list \n Press 9 to 
concatenate two list\n press 0 for exit\n press 99 for goto menu-\t");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        list_traversal(N);
        goto label;
        break;
    case 2:
        N = insert_end(N);
        goto label;
        break;
    case 3:
        N = insert_begin(N);
        goto label;
        break;
    case 4:
        printf("At what posion you want to insert elemnent in linked list\t");
        scanf("%d", &pos);
        N = insert_middle(pos, N);
        goto label;
        break;
    case 5:
        N = delete_head(N);
        goto label;
        break;
    case 6:
        N = delete_end(N);
        goto label;
        break;
    case 7:
        printf("At what posion you want to delete elemnent in linked list\t");
        scanf("%d", &pos);
        N = delete_middle(N, pos);
        goto label;
        break;
    case 8:
        N = reverse(NULL, N);
        goto label;
        break;
    case 9:
        printf("Please create second list to concatinate-\n");
        printf("How many element you want in list 2-\t");
        scanf("%d", &size);
        M = list_creation(size);
        N = concat(N, M);
        goto label;
```

```
            break;
        case 99:
            goto label;
            break;
        case 0:
            exit(0);
            break;
        default:
            printf("enter valid key\n");
            goto label;
        }
    }
```