1)

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

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

void display(struct node *);

int main(){


     struct node *first=(struct node*)malloc(sizeof(struct node));
    struct node *second=(struct node*)malloc(sizeof(struct node));
    struct node *third=(struct node*)malloc(sizeof(struct node));

    first->data=30;
    first->next=second;
    second->data=40;
    second->next=third;
    third->data=50;
    third->next=NULL;

    display(first);


    return 0;
}

void display(struct node *temp){
    while(temp!=NULL){
        printf("%d->",temp->data);
        temp=temp->next;
    }
}
```

2)using recursion

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

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

```
};

void display(struct node *);

int main(){


     struct node *first=(struct node*)malloc(sizeof(struct node));
    struct node *second=(struct node*)malloc(sizeof(struct node));
    struct node *third=(struct node*)malloc(sizeof(struct node));

    first->data=30;
    first->next=second;
    second->data=40;
    second->next=third;
    third->data=50;
    third->next=NULL;

    display(first);//it contains address of first node


    return 0;
}

void display(struct node *temp){
    if(temp!=NULL){
        printf("%d->",temp->data);
        display(temp->next);// contains address of temp->next
    }

}
```

3)reversal of list using recursion

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

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

void display(struct node *);

int main(){


     struct node *first=(struct node*)malloc(sizeof(struct node));
```

```
    struct node *second=(struct node*)malloc(sizeof(struct node));
    struct node *third=(struct node*)malloc(sizeof(struct node));

    first->data=30;
    first->next=second;
    second->data=40;
    second->next=third;
    third->data=50;
    third->next=NULL;

    display(first);//it contains address of first node


    return 0;
}

void display(struct node *temp){
    if(temp!=NULL){
        display(temp->next);// contains address of temp->next

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

}
```

2)

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

struct node {
    int data;
    struct node *next;
} *head = NULL;

// Function prototypes
void Rdisplay(struct node *);
int Rcount(struct node *);
int sum(struct node *);
int Rsum(struct node *);
int max(struct node *);
int Rmax(struct node *);
int Lsearch(struct node *, int);
void insert(int, int);

int main() {
    // Create the initial list
    struct node *first = (struct node *)malloc(sizeof(struct node));
    struct node *second = (struct node *)malloc(sizeof(struct node));
```

```
    struct node *third = (struct node *)malloc(sizeof(struct node));

    first->data = 30;
    first->next = second;
    second->data = 40;
    second->next = third;
    third->data = 50;
    third->next = NULL;

    head = first; // Set head to point to the first node

    Rdisplay(head); // Display the list
    printf("\n");

    int x = Rcount(head);
    printf("Number of nodes: %d\n", x);

    int y = sum(head);
    printf("Sum of elements without recursion: %d\n", y);

    int z = Rsum(head);
    printf("Sum of elements using recursion: %d\n", z);

    int m = max(head);
    printf("Max value in the list: %d\n", m);

    int rmax = Rmax(head);
    printf("Max value in the list using recursion: %d\n", rmax);

    int key_value = 40;
    int search = Lsearch(head, key_value);
    if (search != -1) {
        printf("Value found using linear search: %d\n", search);
    } else {
        printf("Value not found using linear search\n");
    }

    // Insert a new value at position 2
    insert(2, 15);
    printf("List after insertion:\n");
    Rdisplay(head);

    return 0;
}

void Rdisplay(struct node *temp) {
    if (temp == NULL) {
        return;
```

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

int Rcount(struct node *temp) {
    if (temp == NULL) {
        return 0;
    }
    return Rcount(temp->next) + 1;
}

int sum(struct node *temp) {
    int sum = 0;
    while (temp != NULL) {
        sum += temp->data;
        temp = temp->next;
    }
    return sum;
}

int Rsum(struct node *temp) {
    if (temp == NULL) {
        return 0;
    }
    return temp->data + Rsum(temp->next);
}

int max(struct node *temp) {
    int max1 = temp->data;
    while (temp != NULL) {
        if (temp->data > max1) {
            max1 = temp->data;
        }
        temp = temp->next;
    }
    return max1;
}

int Rmax(struct node *temp) {
    if (temp == NULL) {
        return 0;
    }
    int max_in_rest = Rmax(temp->next);
    return (temp->data > max_in_rest) ? temp->data : max_in_rest;
}

int Lsearch(struct node *temp, int key) {
```

```
    while (temp != NULL) {
        if (key == temp->data) {
            return temp->data;
        }
        temp = temp->next;
    }
    return -1; // Key not found
}

void insert(int pos, int value) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = value;

    if (pos == 0) { // Insert at the beginning
        newnode->next = head;
        head = newnode;
    } else {
        struct node *temp = head;
        for (int i = 0; i < pos - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
        if (temp != NULL) {
            newnode->next = temp->next;
            temp->next = newnode;
        } else {
            printf("Invalid position\n");
            free(newnode);
        }
    }
}



void create(int A[],int n){
    struct Node*t,*last;//last mean temp
    head=(struct Node*)malloc(sizeof(struct Node));
    head->data=A[0];
    head->next=NULL;
    last=head;
    for(int i=1;i<n;i++){
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}
```