Q1

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

struct ~~Node~~ link {
        int data;
        struct link * next;
};

typedef struct link node;

node * newNode (int data) {
        node * temp = (node*) malloc (sizeof(node));
        temp -> data = data;
        temp -> next = NULL;
        return temp;
}

void append (node** head, int data) {
        node * temp = newNode (data);
        if (*head == NULL) {
                *head = temp;
                return;
        }
        node * curr = *head;
        while (current -> next != NULL)
                curr = curr -> next;
        curr -> next = temp;
}
```

```
Void printlist (node * head) {

    while (head != NULL) {
        printf ("%d -> ", head -> data);
        head = head -> next;
    }
    printf ("NULL \n");
}

Void rearrange EvenOdd (node ** head) {

    if (*head == NULL || (*head) -> next == NUL
        return;

    node *evenHead = NULL, *evenTail = NULL;
    node *oddHead = NULL, *oddTail = NULL;

    node* curr = *head;
    int pos = 1;

    while (curr != NULL) {
        node * nextNode = curr -> next;
        curr -> next = NULL;

        if (pos % 2 == 0) {
            if (evenHead == NULL) {
                even == evenTail = curr;
            } else {
                evenTail -> next = curr;
                evenTail = curr;
            }
```

```
else {
    if (oddHead == NULL) {
        oddHead = oddTail = curr;
    } else {
        oddTail -> next = curr;
        oddTail = curr;
    }
}
pos++;
curr = nextNode;
}
if (evenHead == NULL) {
    *head = oddhead;
}
evenTail -> next = oddHead
*head = evenHead;
}
int main() {
    struct link * head = NULL;

    append (&head, 1);
    append (&head, 2);
    append (&head, 3);
    append (&head, 4);
    append (&head, 5);
    append (&head, 6);

    printf ("Original List: \n");
    printfList (head);
    rearrangeEvenodd (&head);
    printList (head);
return 0;
}
```

Q2

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

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

struct Node * newNode (int data) {
    struct Node* temp = (struct Node*) malloc(sizeof(
                                    (struct Node));
    temp -> data = data;
    temp -> next = NULL;
    return temp;
}

Void append (struct Node** head, int data) {
    struct Node* temp = newNode (data);
    if (* head == NULL) {
        * head = temp;
        return;
    }

Void printList ( struct Node * head, int k) {
    if ( head == NULL || k <= 1)
        return head;

    struct Node * curr = head;
```

```
struct Node* prevGroupTail = NULL;
struct Node* newHead = NULL;

while (curr != NULL) {

    struct Node* temp = curr;
    int count = 0;

while ( temp != NULL && count < k) {

        temp = temp -> next
        count ++;
}

    if (count < k) {
        if (prevGroupTail != NULL)
            prevGroupTail -> next = curr;
        break;
}

for (int i = 0 ; i < k ; i++) {

        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
}

    if ( newHead == NULL)

        newHead = prev;

}
```

```
int main() {

    struct Node* head = NULL;

    int arr[] = {1,2,3,4,5,6,7,8};
    int n = 8;
    int k = 3;

    for (int i=0 ; i< n ; i++) {
        append (&head, arr[i]);
    }
    printf( "Original List : \n");
    printlist (head);

    printf ("Modified List: \n");

    head = reverseKGroup (head, k);

    printList ( head);

    return 0;

}
```

03

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

struct Request {
        int id;
        char name[50];
        float distance;
        float fare;
        struct Request * next;
};

struct Request* newRequest(int id, char name[], float dist,
                           float fare) {
    struct Request* temp = (struct Request*)(malloc(sizeof(
                                struct Request)));
    temp->id = id;
    strcpy(temp->name, name);
    temp->distance = dist;
    temp->fare = fare;
    temp->next = NULL;

    return temp;
}

void insertRequest(struct Request** head, int id,
                   char name[], float dist, float fare) {
    struct Request* temp = newRequest(id, name, dist,
                                      fare);
```

```
if (*head == NULL || (*head)->distance > dist ||
    ((*head)->distance == dist && (*head)->fare
    < fare)) {
        temp->next = *head;
        *head = temp;
        return;
}

struct Request *curr = *head;
while(curr->next != NULL && (curr->next->
    distance < dist || (curr->next->distance == dist
        && curr->next->fare >= fare)) {

    curr = curr->next;
}

    temp->next = curr->next;
    curr->next = temp;
}

Void display (struct Request *head) {

    if (head == NULL) {

        printf ("ID: %d | Name: %s | Distance: %.2f |
            Fare: %.2f \n", head->id, head->name,
            head->distance, head->fare);

        head = head->next;
    }
}
```

```
Void serveNearest(struct Request ** head){

    if(*head == NULL){

            printf("No requests to serve.\n");
            return;
    }

    struct Request* temp = *head;

    printf("ID: %d | Name: %s | Distance: %.2f |
            Fare: %.2f \n", temp->id, temp->name,
        temp->distance, temp->fare);

    *head = temp->next;

    free(temp);


int countRequests(struct Request* head){
        int count=0;

    while(head != NULL){
                count++;
                head= head->next;

    }
    return count;
}
```

```
Void cancel request ( struct Request ** head, int id) {

    if ( * head == NULL) {
            printf ("No active requests. \n");
            return;
    }

    struct Request * curr = * head;
    struct Request * prev = NULL;

    while ( curr != NULL && curr -> id != id) {

        prev = curr;
        curr = curr -> next;
    }

    if ( curr == NULL) {
        printf ("Not found. \n", );
        return;
    }

    free( curr);
    printf (" Request canceled successfully. \n");

}

int main( ) {
    struct Request * head = NULL;
    int choice;
```

```
while (1) {
    printf("\n  Pickup Request Manager \n");
    printf(" 1. Insert new request \n");
    printf(" 2. Display all request \n");
    printf(" 3. Serve nearest request \n");
    printf(" 4. Count active request \n");
    printf(" 5. Cancel request by ID \n");
    printf("6. Exit \n");
    scanf("%d", &choice);

    if (choice == 1) {
                int id;
                char name[50];
                float dist, fare;
    printf("Enter ID, Name, Distance, fare: ");
    scanf("%d, &id);
    scanf("%s, name);
    scanf("%f", &dist);
    scanf("%f", &fare);

    insertRequest(&head, id, name, dist fare);
    prihtf("Inserted.\n");
}

else if (choice == 2) {
                display(head);
        }

else if (choice == 3) {
                serveNearest(&head);
}
```

```
else if (choice == 4) {
        printf("Total active request : %d \n", countRequest
                                                (head));
    }

else if (choice == 5) {
        int id;
        printf("Enter ID to cancel: ");
        scanf("%d", &id);
        cancelRequest(&head, id);
    }

else if (choice == 6) {
        if (head == NULL)
            printf("All side completed!\n");

        else
            printf("Some pending.\n");
            break;
    }
else {
        printf("Invalid choice.\n");
    }
}

return 0;

}
```