Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
typedef struct node *Node
Node getNode(){
    Node x;
    x = (Node) malloc (sizeof (struct node));
    if (x == NULL){
        printf ("mem.full\n");
        exit(0);
    }
    return x;
}
Node insertRear (Node first, int item) {
    Node temp, current;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
```

```
if (first == NULL) {
        return temp;
}
current = first;
while (current -> next != NULL) {
        current = current -> next;
}
current = first;
while
  current -> next = temp;
  return first;
}

Node insertFront (Node first, int item) {
    Node temp;
    temp = getNode();
    temp -> data = item;
    temp -> next = NULL;
    if (first == NULL) {
        return temp;
    }
    temp -> next = first;
    first = temp
    return first;
}
```

```
Node deleteRear (Node first ){
    Node current, previous;
    if (first == NULL){
        printf ("List is empty cannot delete \n");
        return first ;
    }
    if (first -> next = = NULL){
        printf ("Item deleted is %d \n", first -> data);
        free (first);
        return NULL;
    }
    prev = NULL;
    cur = first;
    while (cur -> next != NULL){
            prev = cur;
            cur = cur -> next;
    }
printf ("Item deleted at rearend is %d \n",
        current -> data);
    free (cur);
    prev -> next = NULL;
    return first;
}
```

```
Node deleteFront(Node first){
    Node temp;
    if (first == NULL){
        pf("List is empty\n");
        return first;
    }
    temp = first;
    temp = temp->next;
    pf("Item deleted is %d\n", first->data);
    free(first);
    return temp;
}
Node insertPos(int item, int pos, Node first){
    Node temp, prev, cur;
    int count;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if (first == NULL && pos == 1){
        return temp;
    }
    if (first == NULL){
        pf("Invalid pos\n");
        return first;
    }
```

```
if (pos == 1) {
    temp -> next = first;
    return temp;
}

count = 1;
prev = NULL;
cur = first;
while (cur != NULL && count != pos) {
    prev = cur;
    cur = cur -> next;
    count++;
}
if (count == pos) {
    prev -> next = temp;
    temp -> next = cur;
    return first;
}
Pf("Inv pos \n");
ret first;
}
```

```
Node concat (Node first, Node second) {
    Node cur;
    if (first == NULL) {
        return second;
    }
    if (second == NULL) {
        return first
    }
    cur = first;
    while (cur -> next != NULL) {
        cur = cur -> next;
    }
    cur -> next = second;
    return first;
}
Node swap (Node a, Node b) {
    int temp = a -> data;
    a -> data = b -> data;
    b -> data = temp;
}
```

```
void sort (Node first ){
    int swapped, i;
    Node cur
    if (first == NULL){
        pf("empty\n");
        return;
    }
    do {
        swapped = 0;
        cur = first;
        while (cur -> next != NULL) {
            if (cur -> data > cur -> next -> data){
                swap (cur, cur -> next);
                swapped = 1;
            }
            cur = cur -> next;
        }
    }
    while (swapped);
}
```

```
Node reverse (Node first) {
    Node cur, temp;
    cur = NULL;
    while (first != NULL) {
        temp = first;
        first = first->next;
        temp->next = cur;
        cur = temp;
    }
    return cur;
}

void display (Node first) {
    Node temp;
    if (first == NULL) {
        Pf ("empty\n");
        return;
    }
for (temp= first ; temp != NULL ; temp = temp->
        Pf ("%d ", temp->data);
}
Pf ("\n");
}
```

```
void main () {
    int item, choice, flag=1, n, i;
    Node first = NULL;
    Node a, b;
  while (flag ==1){
   pf ("\n1. Insert Front \n2. Insert Rear \n
       3. Delete Rear \n 4. Delete Rear \n
       5. Order list \n 6. Insert at \n 7. Delete a
       8. Reverse \n 9. Concat \n 10. Sort \n 11.
       Display \n 12. Exit \n");
    pf (" Enter your choice \n");
    sconf ("%d", &choice);
  switch (choice) {
  case 1; pf (" Enter the item : \n");
          sconf ("%d", &item);
          first = insert Front (first, item);
          break;
 case 2: pf (" Enter the item: \n");
         sconf ("%d", &item);
         first= insert Rear (first, item);
         break;
```

```
case 3: first = deleteFront(first);
        break;
case 4: first = deleteRear(first);
        break;

case 5: first = reverse(first);
        display(first);
        break;
        pf("enter item: \n");
        scanf("%d", &item);
        first = orderList(first, item);
        break;

case 6: printf("Enter item: \n");
        scanf("%d", &item);
        printf("Enter pos: \n");
        scanf("%d", &pos);
        first = insert_position(item, pos,
                                first);
case 7: pf("Enter pos:");
        scanf("%d", &pos);
        first = deleteposition(pos, first);
        break;
case 8: first = reverse(first);
        display(first);
        break;
```

```
case 9: {
        pf("\n enter total nodes in list 1:\n");
        sconf("%d", &n);
       a = NULL;
       for(i=0; i<n; i++){
          pf("Enter item:\n");
          sconf("%d", &item);
        a = insertRear(a, item);
      }
      pf("Enter total nodes in list 2:\n");
       sconf("%d", &n);
       b = NULL;
      for(i=0; i<n; i++) {
       pf("Enter the item:");
        sconf("%d", &item);
       b = insertRear(b, item);
   }
   a = concat(a, b);
   display(a);
  break;
}
```

```
case 10: sort(first);
         display(first);
         break;

case 11: display(first);
         break
default: exit(0)

  }

 }

}
```