1.main.c

```
#include "header.h"

int main(){
   CLL L1;
   init_CLL(&L1);
   insert_beg(&L1, 5);
   insert_beg(&L1, 10);
   insert_beg(&L1, 15);
   insert_end(&L1, 20);
   insert_end(&L1, 25);
   insert_end(&L1, 30);
   display(&L1);
   insert_pos(&L1, 50, 4);
   display(&L1);
   sort(&L1);
   printf("After Sorting: ");
   display(&L1);
   remove_beg(&L1);
   printf("After removing first element: ");
   display(&L1);
   remove_end(&L1);
   printf("After removing last element: ");
   display(&L1);
   remove_pos(&L1, 3);
   printf("After removing element from position 3: ");
   display(&L1);
   return 0;
}
```

2.header.h

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

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

typedef struct CLL{
   node *front, *rear;
}CLL;

void init_CLL(CLL *l);
int isEmpty(CLL *l);
void insert_beg(CLL *l, int d);
void insert_end(CLL *l, int d);
void insert_pos(CLL *l, int d, int pos);
void remove_beg(CLL *l);
void remove_end(CLL *l);
void remove_pos(CLL *l, int pos);
void sort(CLL *l);
void display(CLL *l);
```

3.logic.c

```
#include "header.h"


void init_CLL(CLL *l){
   l->front = NULL;
   l->rear = NULL;
}
int isEmpty(CLL *l){
   if(l -> front == NULL && l -> rear == NULL){
      return 1;
   }
   return 0;
}
void insert_beg(CLL *l, int d){
   node *newnode = (node *)malloc(sizeof(node));
   newnode -> data = d;
   newnode -> next = NULL;
   if(isEmpty(l)){
      l -> front = newnode;
      l -> rear = newnode;
      newnode -> next = l -> front;
   }
   else{
      newnode -> next = l -> front;
      l -> rear -> next = newnode;
      l -> front = newnode;
   }
}
void insert_end(CLL *l, int d){
   node *newnode = (node *)malloc(sizeof(node));
   newnode -> data = d;
```

```
    newnode -> next = NULL;
    if(isEmpty(l)){
        l -> front = newnode;
        l -> rear = newnode;
        newnode -> next = l -> front;
    }
    else{
        l -> rear -> next = newnode;
        l -> rear = newnode;
        newnode -> next = l -> front;
    }
}
void insert_pos(CLL *l, int d, int pos){
    node *newnode = (node *)malloc(sizeof(node));
    newnode -> data = d;
    newnode -> next = NULL;
    if(pos == 1){
        insert_beg(l, d);
        return;
    }
    else{
        node *temp = l -> front;
        int i = 1;
        while(i < pos - 1 && temp -> next != l -> front){
            temp = temp -> next;
            i++;
        }
        if(i == pos - 1){
            newnode -> next = temp -> next;
            temp -> next = newnode;
            if(newnode -> next == l -> front){
```

```
            l -> rear = newnode;
          }
        }
        else{
          printf("position is out of range\n");
          free(newnode);
        }
    }

}
void remove_beg(CLL *l){
    if(isEmpty(l)){
        printf("List is empty");
        return;
    }
    node *newnode = l -> front;
    if(l -> front == l -> rear){
        l -> front = NULL;
        l -> rear = NULL;
    }
    else{
        l -> front = newnode -> next;
        l -> rear -> next = l -> front;
    }
}
void remove_end(CLL *l){
    if(isEmpty(l)){
        printf("List is empty");
        return;
    }
    if(l -> front == l -> rear){
```

```
        l -> front = NULL;
        l -> rear = NULL;
    }
    else{
        node *temp = l -> front;
        while(temp -> next != l -> rear){
            temp = temp -> next;
        }
        l -> rear = temp;
        temp -> next = l -> front;
    }

}
void remove_pos(CLL *l, int pos){
    if(isEmpty(l)){
        printf("List is empty");
        return;
    }
    if(pos == 1){
        remove_beg(l);
        return;
    }
    else{
        node *temp = l -> front;
        int i = 1;
        while(i < pos - 1 && temp -> next != l -> front){
            temp = temp -> next;
            i++;
        }
        if(i == pos - 1 && temp -> next != l -> front){
            node *q = temp -> next;
```

```
            if(q == l -> rear){
                l -> rear = temp;
            }
            temp -> next = q -> next;
            free(q);
        }
        else{
            printf("Position is out of range\n");
        }
    }

}
void sort(CLL *l){
    if(isEmpty(l)){
        return;
    }
    int swap;
    node *temp;
    do{
        swap = 0;
        temp = l -> front;
        while(temp -> next != l -> front){
            if(temp -> data > temp -> next -> data){
                int t = temp -> data;
                temp -> data = temp -> next -> data;
                temp -> next -> data = t;
                swap = 1;
            }
            temp = temp -> next;
        }
        l -> rear = temp;
```

```
    }while(swap);
}

// void sort(CLL *l){
//     if(isEmpty(l)){
//         return;
//     }
//     int temp, sorted = 0;
//     node *q = NULL;
//     while(!sorted){
//         sorted = 1;
//         node *p = l -> front;
//         do{
//             q = p -> next;

//         }
//     }
// }
void display(CLL *l){
    if(isEmpty(l)){
        printf("List is empty\n");
        return;
    }
    node *temp = l -> front;
    do {
        printf("%d -> ", temp -> data);
        temp = temp -> next;
    }while(temp != l -> front);
    printf("(back to front)\n");
}
```

OUTPUT:

```
tanis@Tanishq MINGW64 /d/COEP/DSA/LabWork/CLL
$ gcc -Wall main.c logic.c

tanis@Tanishq MINGW64 /d/COEP/DSA/LabWork/CLL
$ ./a
15 -> 10 -> 5 -> 20 -> 25 -> 30 -> (back to front)
15 -> 10 -> 5 -> 50 -> 20 -> 25 -> 30 -> (back to front)
After Sorting: 5 -> 10 -> 15 -> 20 -> 25 -> 30 -> 50 -> (back to front)
After removing first element: 10 -> 15 -> 20 -> 25 -> 30 -> 50 -> (back to front)
After removing last element: 10 -> 15 -> 20 -> 25 -> 30 -> (back to front)
After removing element from position 3: 10 -> 15 -> 25 -> 30 -> (back to front)
```