Exercise 1
A)

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

/*This is the struct that will construct our list*/
struct Node{
   int num; /*Here is the key*/
   struct Node *next;
};


/* This pushes a node with node->data=num at the begging of list*/
void listPush(struct Node** l,int num){
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->num=num;

   node->next=(*l);
   (*l)=node;
}


/*This prints the list so we can see the results*/
void listPrint(struct Node* l){
   printf("Head");
   while(l!=NULL){
      printf("->");
      printf("%d",l->num);
      l = l->next;
   }
   printf("->NULL\n");
}
```

```
/*This removes all nodes that have node->data == k */
void RemoveKey(struct Node**L,int k){
   struct Node *current=(*L),*prev;
   if(current==NULL){
      printf("The list is empty\n");
      return;
   }
   while(current!=NULL && current->num==k){
      (*L)=current->next;
      free(current);
      current=(*L);
   }
   while(current!=NULL){
      while(current!=NULL && current->num!=k){
         prev = current;
         current = current->next;
      }
      if(current==NULL){
         return;
      }
      if(current->next==NULL){
         free(current);
         prev->next=NULL;
         return;
      }
      prev->next=current->next;
      free(current);
      current=prev->next;
   }
}
```

```
/* This is Question A from exercise 1*/
int main(){
   /*This is the head of the list*/
   struct Node *head=NULL;
   /*We create a list Head->4->5->4->3->2->4
     This list tests all cornern cases, if key is found at the begging, at the midle, at the end.
   */
   listPush(&head,4);
   listPush(&head,2);
   listPush(&head,3);
   listPush(&head,4);
   listPush(&head,5);
   listPush(&head,4);
   listPrint(head);
   /*This will delete all nodes with key's 4*/
   int key = 4;
   RemoveKey(&head,key);
   /*Here we can see that all the 4's have been removed from the list.*/
   printf("Removing all nodes with data = %d\n",key );
   listPrint(head);
   /*Test custom input here */

}
```

B)

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

/*This is the struct that will construct our list*/
struct Node{
   int num; /*Here is the key*/
   struct Node *next;
};

/* This pushes a node with node->data=num at the begging of list*/
void listPush(struct Node** l,int num){
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->num=num;

   node->next=(*l);
   (*l)=node;
}

/*This prints the list so we can see the results*/
void listPrint(struct Node* l){
   printf("Head");
   while(l != NULL){
      printf("->");
      printf("%d",l->num);
      l = l->next;
   }
   printf("->NULL\n");
}
/*_____*/
/*Now we need to implement a stack*/
```

```
/*Stack that is implemented with linked lists*/
struct stack{
   struct Node *node;
   struct stack *next;
};
/*This pop's a node from stack*/
struct Node* pop(struct stack** L){
   struct stack *tmp = (*L);
   if(tmp==NULL){
      printf("Stack is empty\n");
      exit(-1);
   }
   (*L)=(*L)->next;
   struct Node* node = tmp->node;
   free(tmp);
   return node;
}
/*This pushed a node at the top of the stack*/
void push(struct stack** sp,struct Node* pointer){
   struct stack* node = (struct stack*)malloc(sizeof(struct stack));
   node->node=pointer;

   node->next=(*sp);
   (*sp)=node;
}
/*Exercise1b is implemented in three phases we have to iterate at
the end of the list while pushing all the nodes at the stack,
then read the data from the last node and calculate which node
should be deleted,
then we pop from the stack until we reach the desired node and
delete it.
I made this implementation because Mr's Fatourou said to iterate
the list 1 time. If we iterate the list 2 times
one to read the last node and two to delete the desired node we
have time complexity of 2n -> O(n).
```

```
The way i implement my algorith the time complexity is 2n too -> O(n). But my algorithm takes up more space
because we save at the stack. I would do it with 2 iterations but i think Mr's Fatourou said to do it with 1
and secondly i wanted to try this implementation with the stack because it is in one of Mr's Fatourou examples.
*/
void exercise1b(struct Node **L){
   int n = 0; // Counter
   struct Node* node = (*L);
   struct stack* stack = NULL; // Our stack
   // First Phase iterate to last node.
   while(node!=NULL){
      push(&stack,node);
      node=node->next;
      n++;
   }
   // Second Phase calculate how many nodes we have to pop from stack.
   struct Node* current,*prev;
   current = pop(&stack);
   int num = current->num;
   int index;
   if(n-num > 0){
      index = num;
   }else{
      index = num%n;
   }

   // Pop the nodes from stack
   if(index==0){
      prev = pop(&stack);
      prev->next = NULL;
      free(current);
      return;
```

```
    }

    int tmp = index;

    while(tmp--){
        current = pop(&stack);
    }
    if(index == n-1){
        (*L) = current->next;
        free(current);
        return;
    }
    prev = pop(&stack);
    // Third Phase delete the node from the stack
    prev->next=current->next;
    free(current);
}


int main(){
    struct Node *head=NULL;
    /*We are going to test all corner cases*/

    /*First Case num>n*/
    listPush(&head,5);
    listPush(&head,2);
    listPush(&head,3);
    listPush(&head,1);
    listPrint(head);
    printf("First case num > n\n");
    printf("Last node is 5 size is 4 so we have to delete (5 mod 4) =1,
deleting the previous node.\n");
    exercise1b(&head);
    listPrint(head);
```

```
    /*Second Case*/
    /*num%n = 5%4 = 1. So we have to delete the previous node of
the last one*/
    head=NULL;
    printf("\n");
    listPush(&head,8);
    listPush(&head,2);
    listPush(&head,3);
    listPush(&head,1);
    listPrint(head);
    printf("Second case num > n -> (num mod n) =0\n");
    printf("In this case (8 mod 4) =0 so we delete the last node\n");
    exercise1b(&head);
    listPrint(head);


    /*Third Case*/
    /*n > num so num=n-1 the first node*/
    head=NULL;
    printf("\n");
    listPush(&head,3);
    listPush(&head,2);
    listPush(&head,3);
    listPush(&head,1);
    listPrint(head);
    printf("Third case n>num so we delete the first node because num
= n-1 \n");
    exercise1b(&head);
    listPrint(head);
```

```
    /*Fourth Case*/
    /*n>num so we delete the node that is num nodes before last
node*/
    head=NULL;
    printf("\n");
    listPush(&head,2);
    listPush(&head,2);
    listPush(&head,3);
    listPush(&head,1);
    listPrint(head);
    printf("Last node's data is 2 so we delete the node that is 2 nodes
before. Node with data = 3\n");
    exercise1b(&head);
    listPrint(head);
    /*Test Custom input Here */
}
```

C)

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

//struct Node *sentinel;

struct Node{
   int num;
   struct Node *next;
} *sentinel;

/*Inserting a node at the begging of the list if no other node has num
= k*/
void Insert(struct Node **L,int k){
   struct Node* node;
   for(node = *L;
      node != sentinel;
      node = node->next)
   {
      if(node->num==k){
         printf("Node with num=%d already exists\n",k);
         return;
      }
   }
   node = (struct Node*)malloc(sizeof(struct Node));
   node->num = k;
   node->next = *L;
   *L=node;

}
```

```
/*Deletes the node that has num=k if it exists*/
void Delete(struct Node **L,int k){
   struct Node* current,*prev;
   for(current = *L;
      current != sentinel;
      current = current->next)
   {
      if(current->num==k){
         if(current == *L){
            *L=current->next;
            free(current);
         }else{
            prev->next=current->next;
            free(current);
         }
         return;
      }
      prev=current;
   }
}

/*Prints the list*/
void Sprint(struct Node *l){
   printf("Head");
   struct Node *node;
   for(node = l;
      node != sentinel;
      node = node->next)
   {
      printf("->");
      printf("%d",node->num);
   }
   printf("->NULL\n");
```

```
}


int main(){
   struct Node *sentinelHead; /*Head of the list*/
   sentinel = (struct Node*)malloc(sizeof(struct Node));/*Sentinel of
the list*/
   sentinelHead = sentinel;
   /*Insert some values*/
   Insert(&sentinelHead,1);
   Insert(&sentinelHead,2);
   Insert(&sentinelHead,3);
   Insert(&sentinelHead,4);
   Insert(&sentinelHead,6);
   Insert(&sentinelHead,4);
   Insert(&sentinelHead,1);
   /*We inserted two duplicate values so we are going to get two
errors saying 4 and 1 already exists*/
   Sprint(sentinelHead);
   /*we delete a value that exists*/
   Delete(&sentinelHead,1);
   Sprint(sentinelHead);
   /*We delete a value that doesnt exist */
   Delete(&sentinelHead,10);
   Sprint(sentinelHead);
   /*We delete the whole list*/
   Delete(&sentinelHead,2);
   Delete(&sentinelHead,3);
   Delete(&sentinelHead,4);
   Delete(&sentinelHead,6);
   Sprint(sentinelHead);
}
```

Exercise 2
A)

```
Queue* getQueue(){
   Queue *Q;
   MakeEmptyQueue(&Q);
   char c;
   while(c = getchar() != '\0'){
      enQueue(&Q,c);
   }
   return Q;
}

boolean CompareWithQueue(int size){
   Queue *q = getQueue();
   char c,tmp;
   for(int i=0;i<size/2;i++){
      c = deQueue(&q);
      enQueue(&q,c);
      for(int j=0;j<size/2;j++){
         tmp = deQueue(&q);
         enQueue(&q,tmp);
      }
      if(c != tmp) return 0;
   }
   return 1;
}
```

B)

```
void RecursionCheck(int size,struct Stack* stack,struct Stack*
stack2){
   char c;
   if(size>0){
      c = pop(&stack);
      RecursionCheck(size-1,&stack,&stack2);
      push(&stack2,c);
   }
   return;
}

boolean CompareWithStacks()
{
   int size = 0;
   char c;

   Stack* stack;
   Stack* stack2;
   MakeEmptyStack(&stack);
   MakeEmptyStack(&stack2);

   while(c = getchar() != '\n'){
      push(&stack,c);
      size++;
   }

   return check(size/2,stack,stack2);
}
```

Exercise 3

```
struct Node* MergeSortedLists(struct Node* head, struct Node*
head2) {
   static struct Node* head3 = NULL;

   if(!head){
      listPush(&head3,head2->num);
      return head3;
   }else if(!head2){
      listPush(&head3,head->num);
      return head3;
   }

   if(head->num>head2->num){
      listPush(&head3,head->num);
      head3 = MergeSortedLists(head->next,head2);
   }else{
      listPush(&head3,head2->num);
      head3 = MergeSortedLists(head,head2->next);
   }
   return head3;
}
```

We have two pointers at the begging of the two lists. We must first check if the lists are not NULL(Empty).
Then we compare the values of each list and the one that has the most value is going to be assigned to it's
next pointer the return value of the recursive algorithm(That means we call the algoritmh again but we give as the parameter head->next to the one that had the least data).That way the algorithm run's for each iteration and when
NULL is found the tail of the list becomes NULL.
The time complexity is 0(n) because we only iterate n times for each node the two lists have compined.The two lists
at the start have n/2 nodes and the resulting lists has n nodes.In every iteration we assign a node so we have n
iterations therefore we have order n complexity.

T(0) = 2
T(n) = T(n-1)+4
T(1) = 6

T(n) = T(n-1)+4 = T(n-2)+4+4 = T(n-3)+4+4+4 = T(n-1*n)+4*n = T(0)+4*n = 2+4*n
If we asume c >= 4 the we can say c*n>=T(n) So we can say that the algorithm has time complexity

of O(n).

Exercise 1

A)

```
D:\csd\hy240\csd4406\Ex1>a.exe
Head->4->5->4->3->2->4->NULL
Removing all nodes with data = 4
Head->5->3->2->NULL
```

B)

```
D:\csd\hy240\csd4406\Ex1>a.exe
Head->1->3->2->5->NULL
First case num > n
Last node is 5 size is 4 so we have to delete (5 mod 4) =1, deleting the previous node.
Head->1->3->5->NULL

Head->1->3->2->8->NULL
Second case num > n -> (num mod n) =0
In this case (8 mod 4) =0 so we delete the last node
Head->1->3->2->NULL

Head->1->3->2->3->NULL
Third case n>num so we delete the first node because num = n-1
Head->3->2->3->NULL

Head->1->3->2->2->NULL
Last node's data is 2 so we delete the node that is 2 nodes before. Node with data = 3
Head->1->2->2->NULL
```

C)

```
D:\csd\hy240\csd4406\Ex1>a.exe
Node with num=4 already exists
Node with num=1 already exists
Head->6->4->3->2->1->NULL
Head->6->4->3->2->NULL
Head->6->4->3->2->NULL
Head->NULL
```

This photos are from exrcise 1 if we run all the mains