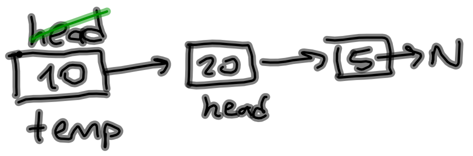
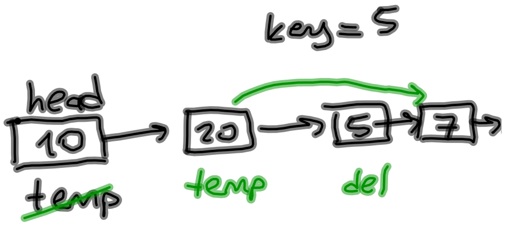
Counting recursively

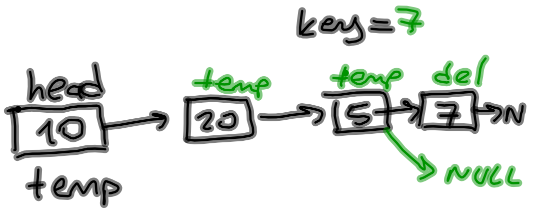
```
int countrec(struct node *head){
        if (head==NULL)
            return 0;
        return 1+countrec(head->next);
}
```

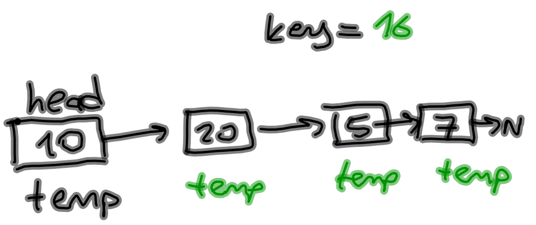
Delete a node with a given key

```
struct node *del(struct node *head, int key){
    struct node *temp=head;
    if(head!=NULL){
        if(head->data==key){
            head=head->next;
            free(temp);
        }else{
            while(temp->next!=NULL && temp->next->data!=key)
                temp=temp->next;
            if(temp->next==NULL)
                return head;
            struct node *del=temp->next;
            temp->next=del->next;
            free(del);
        }
    }
    return head;
}
```

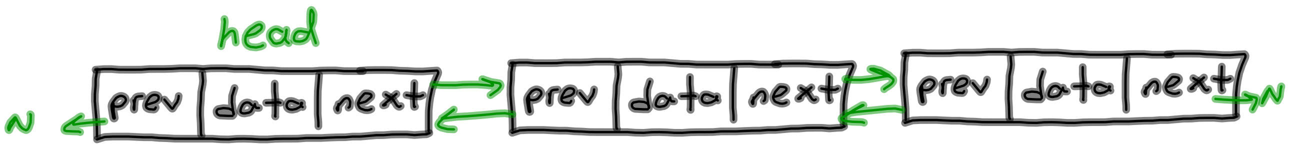








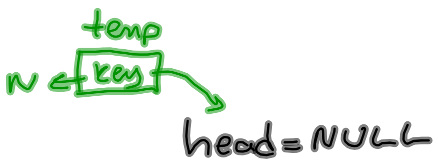
Double Linked List

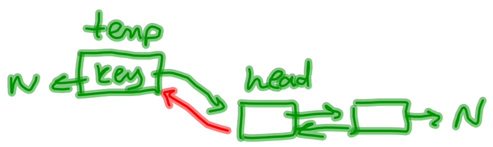


```
struct node{
    int data;
    struct node*next;
    struct node *prev;
}
```

Inserting a node in front of the double Linked list

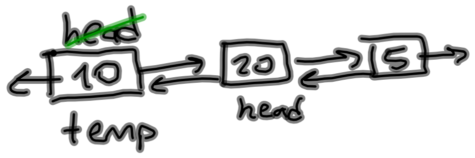
```
struct node *addfront(struct node *head, int key){
    struct node *temp= .... malloc....;
    temp->data= key
    temp->next= head;
    temp->prev= NULL;
    if ( head== NULL)
            head= temp;
    else{
        head->prev=temp;
        head= temp;
    }
    return head;
}
```

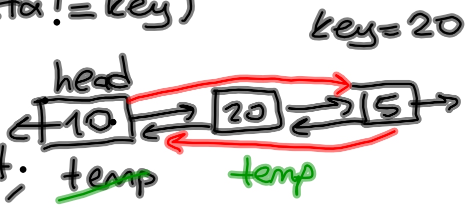


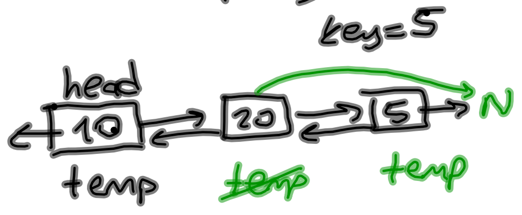


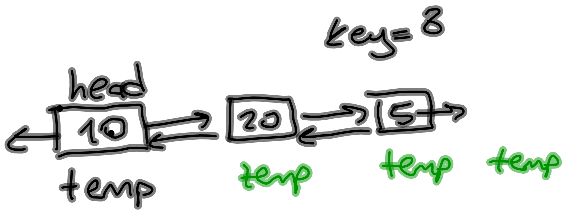
Delete a node with a given key in a DLL

```
struct node *del(struct node *head, int key){
    struct node *temp=head;
    if(head!=NULL){

        if(head->data==key){
            head=head->next;
            head->prev=NULL;
            free(temp);
        }else{
            while(temp!=NULL && temp->data!=key)
                temp=temp->next;
            if(temp!=NULL) return head;
            temp->prev->next=temp->next;
            if(temp->next!=NULL)
                temp->next->prev=temp->prev;

            free(temp);

        }
    }
    return head;
}
```



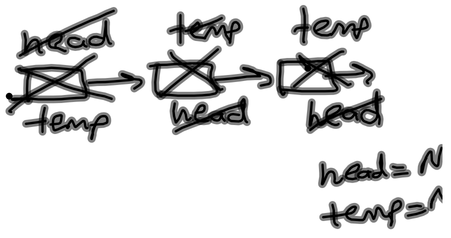






Destroy an SLL

```
struct node *destroy(struct node *head){
    struct node *temp=head;
    while(head!=NULL){
        head=head->next;
        free(temp);
        temp=head;
    }
    return head;
}
```

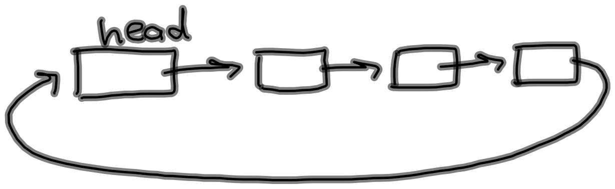


HW: Write the destroy function recursively.

DLLs compared to SLLs

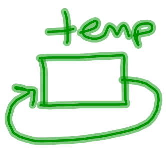
- Advantages:
  - Can be traversed in either direction (may be essential for some programs)
  - Some operations, such as deletion and inserting before a node, become easier
- Disadvantages:
  - Requires more space
  - List manipulations are slower (because more links must be changed)
  - Greater chance of having bugs (because more links must be manipulated)

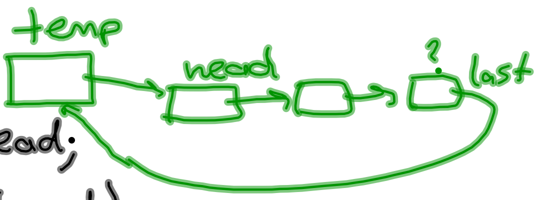
Circular Linked List



Inserting a node in front of the CLL

```
struct node *addhead(struct node *head, int key){
    struct node *temp= .... malloc.....;
    temp->data= key
     if (head== NULL){
          temp->next= temp;
          head=temp;
     }else{
          temp->next=head;
          struct node *last= head;
          while(last->next!= head)
               last= last->next;
          last->next=temp
          head= temp;
     }
     return head;
}
```





Inserting a node at the end of the CLL

we must delete head=temp statements