Detect-Make-Remove Cycle Linked List

```
/////////////// make a cycle function //////////////
void makeCycle(Node* &head, int pos){

    Node* temp=head;
    Node* startNode;// points to start of cycle
    int count=1;
    while(temp->next!=NULL){
        if(count==pos){
            startNode=temp;
        }
        temp=temp->next;
        count++;
    }
    temp->next=startNode;// last node points to start of the cycle
}
///////////////////// DETECT CYCLE FLOYD'S ALGORITHM/////////////////
bool detectCycle(Node* &head){

    Node* fast=head; //hare
    Node* slow=head;  // tortoise

    while(fast!=NULL && fast->next!=NULL){ // if cycle is not present fast
will be first to traverse the list
        slow=slow->next;// moves one step at a time
        fast=fast->next->next;//moves two step at a time

        if(fast==slow){// fast and slow both poiting to same node means there
is a cycle
            return true;}}
        return false;}
////////////////Remove cycle/////////////////
void removeCycle(Node* &head){
    // bring the hare and tortoise to point to the same node then hare should
point to the head node and then again both should take the steps ahead
    Node* fast=head; //hare
    Node* slow=head;  // tortoise
    do{
        slow=slow->next;
        fast=fast->next->next;
    }while(slow!=fast);
    fast=head;
    while(slow->next!=fast->next){
        slow=slow->next;
        fast=fast->next;
    }
    slow->next=NULL;
}
```

Append last K nodes at the start of Linked list

```
////////// function to find Length of Linked list///////
int length(Node* head){
    Node*temp=head;
    int l=0;
    while(temp!=NULL){
        l++;
        temp=temp->next;
    }
    return l;
}


///////////// append last k nodes to start//////

Node* knodes(Node* &head, int k){
    Node*tail=head;
    Node*newHead;
    Node*newTail;
    int leng=length(head);
     k=k%leng; // if k>leng, to make leng-k>0
    int count=1;
    while(tail->next!=NULL){
        if(count==(leng-k)){ // after which our k nodes start
            newTail=tail;
        }
        if(count==leng-k+1){
            newHead=tail;
        }
        tail=tail->next;
        count++;
    }
    newTail->next=NULL;
    tail->next=head;
    head=newHead;
    return newHead;
}
```