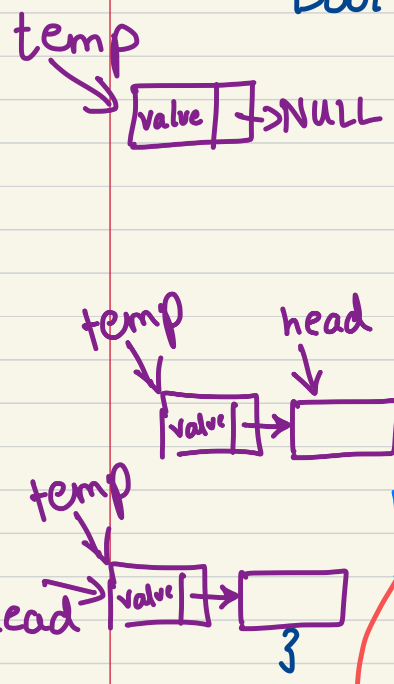
APS 105 Lecture 31 Notes

Last lecture: forming a linked list including some operations like createNode and insertNode at front of the list

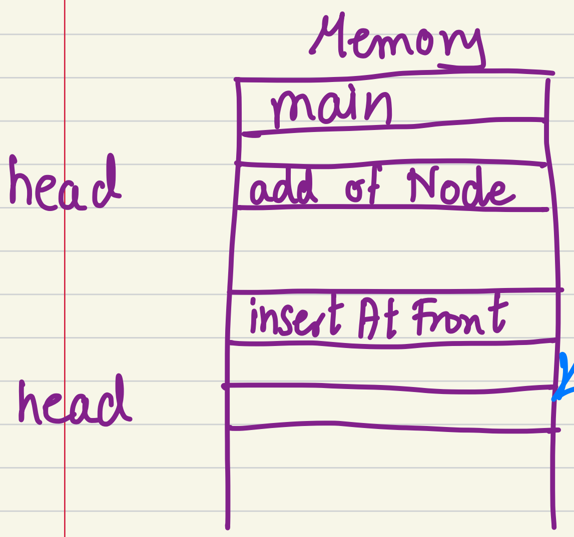
Today: More operations on a linked list

Recap:

```
bool insertAtFront(Node *head, int value){
    Node * temp = createNode (value);
    if (temp == NULL) {  → not enough heap memory available
        return false;
    }
    temp -> next = head;
    head = temp; ??
    return true;
}
```



What we're changing here is value of head



Changes value of head in insertAtFront, but not main()

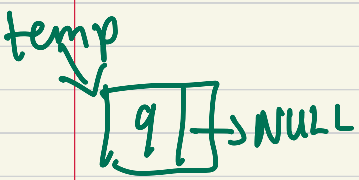
```
typecast struct Nstruct{
        int data;
        struct Nstruct * next;
}Node;

int main(){
        Node * head = NULL;
        insertAtFront ( head, 2);
        will return with head = NULL; as it was passed by value!
        return 0;
}
```

Solution #1: Pass a pointer to head, instead of head (very confusing with double pointers)

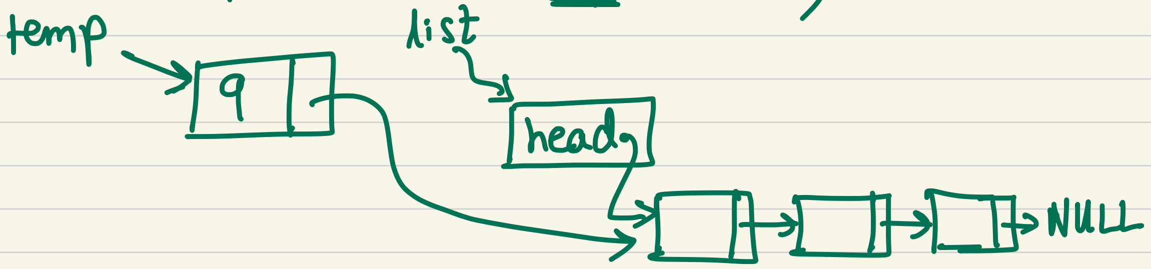
Solution #2:

```
typedef struct list{
        Node * head;

}LinkedList;
bool insertAtFront ( LinkedList *list, int value);
                     pointer to list
                        in main
int main (){
    LinkedList list;
    list.head = NULL;
    insertAtFront(&list, 9);
    return 0;

}
```

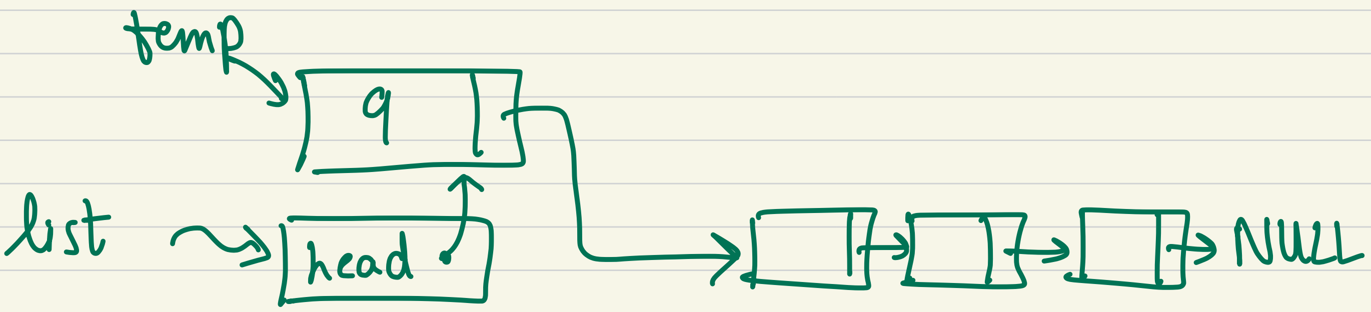
```
bool insertAtFront(LinkedList *list, int value){
```



```
    Node* temp = createNode (value);
    if (temp == NULL){
            return false;
    }

    temp -> next = list -> head;
```



```
    list -> head = temp;
```



```
}

int main (){
    LinkedList list;
    list.head = NULL;
}
```

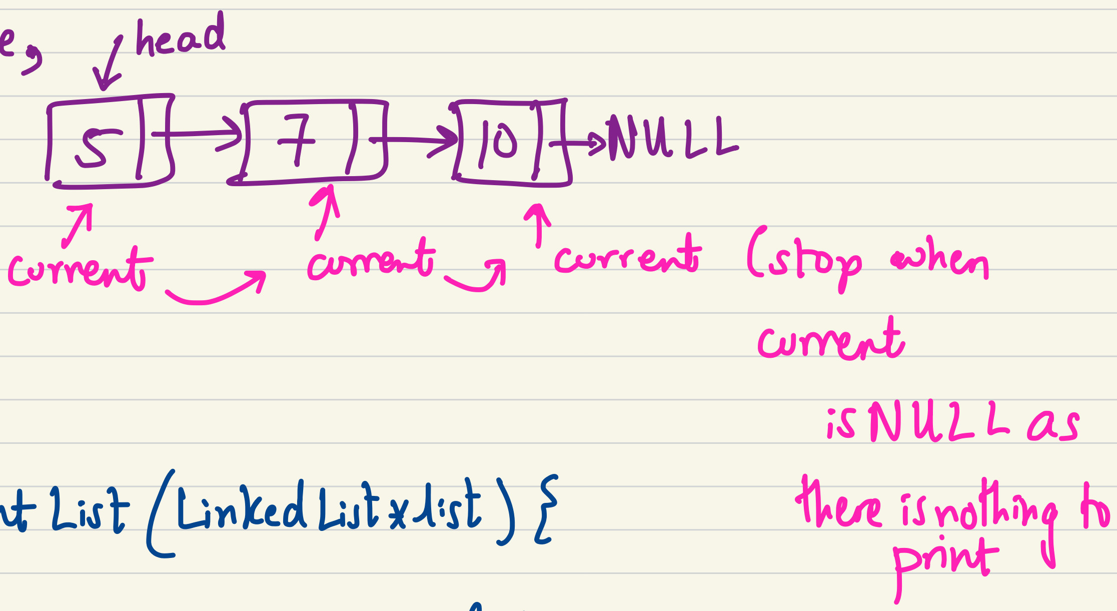
$\Longrightarrow$ Not elegant! let's make a function that initializes an empty list

```
void initList(LinkedList *list){
        list -> head = NULL;

}
```

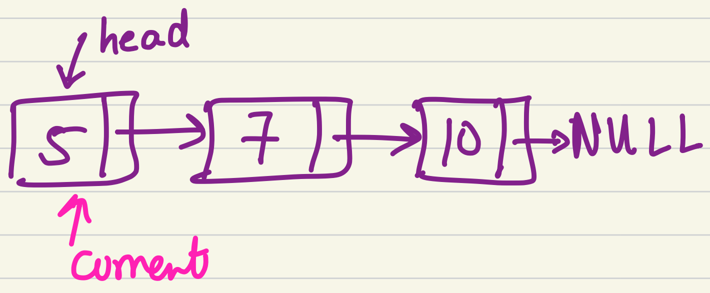
Let's have a function to print elements in a linked list

we have to iterate the linked list 1 node at a time and print its value

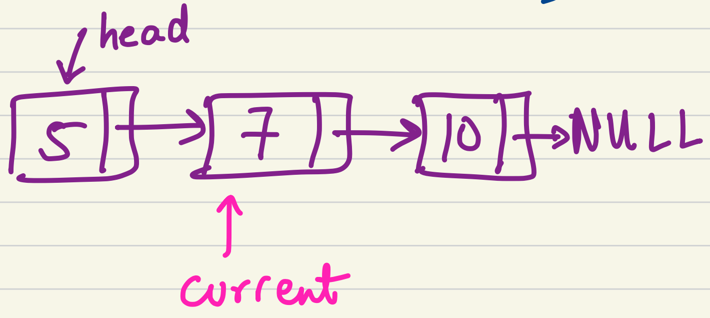
Example,



```
void printList(LinkedList *list) {
        Node * current = list -> head;
```



```
        while(current != NULL) {
                printf("%d \n", current -> data);

                current = current -> next;
```



```
        }
}
```

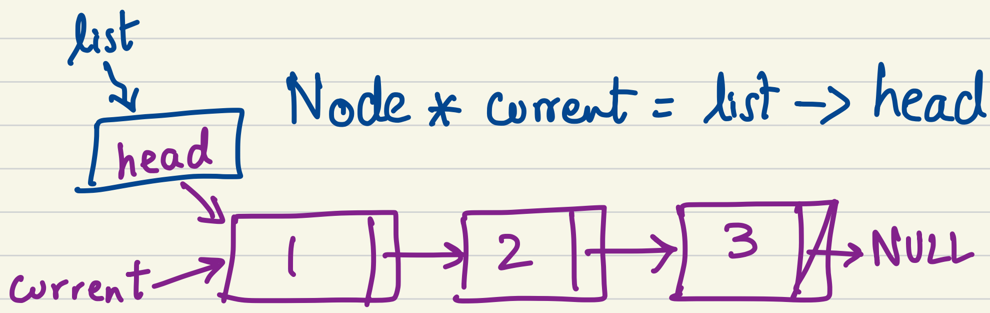
Need to think of special cases:

1. If the list is empty, does the function work
2. If the list has 1 node, does the function work

Let's write a function that looks for a node within the linked list:

Pass a value to look in list and list
& return a pointer to the node that has the value.

```
Node* findFirstNode (LinkedList * list, int value) {
```



```
    Node * current = list -> head;

    while(current != NULL ) {
        if ( current ->data == value)
            return current;
        else {
            current = current ->next;
        }
    }
    return NULL;   → if I couldn't find the value
}
```

Check special cases:

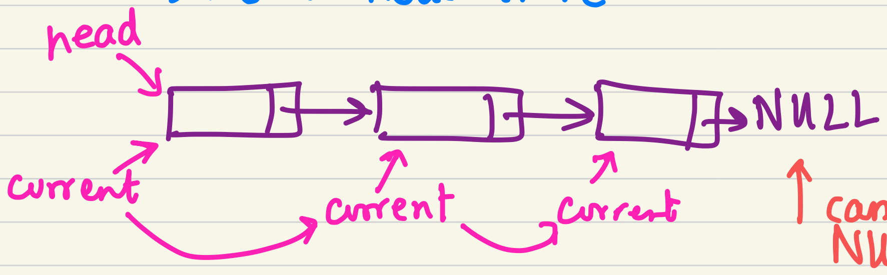
1. If empty list
2. If only 1 node
3. If not found

What if we want to insert at tail of list!

→ Need to traverse list

→ reach last node

→ insert node there



Can insert if current→next is NULL

→ current→next = createNode(9);

```
bool insertAtBack(LinkedList *list, int value ){

        Node * current = head;
        while ( current -> next  != NULL){

                current = current ->next;
        }
here
current ->next is NULL →
        current ->next = createNode(value);
        if (current ->next != NULL) return true;
        else   return false;
}
```

Check special case:

① If list is empty

current -> next will get us segmentation fault since current is NULL

Special case if list is empty

General Case

here current->next is NULL

```
bool insertAtBack(LinkedList *list, int value){
    if(list->head == NULL){
        list->head = creatNode(value);
        return (list->head != NULL);
                OR
        return insertAtFront(list, value);
    }
    Node * current = head;
    while (current->next != NULL){
        current = current->next;
    }
    current->next = createNode(value);
    if (current->next != NULL) return true;
    else return false;
}
```

② Works if there is only 1 node on list