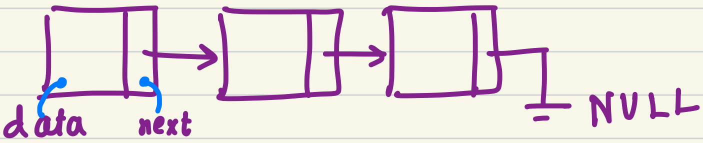
APS 105 Lecture 30 Notes

Last time: We introduce a new list type apart from array, linked list

Today: we implement functions to do operations on linked lists.

Recap: A linked list consists of user-defined Nodes that are linked together with pointers

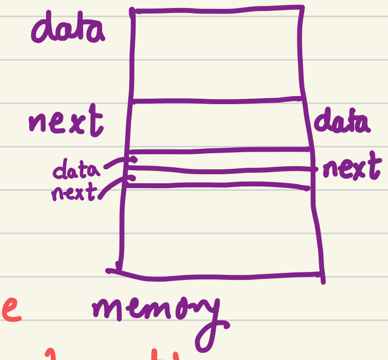


To create a linked list, we first need to define a Node.

```
typedef struct NStruct{
        int data;
        struct Nstruct * next;
} Node;
```

① if this was struct Nstruct next, "Compile-time error" Why?

It would be a never-ending recursion in allocating space for a node.



② if it was Node * next, "Compile-time error", since the alias node doesn't exist yet!

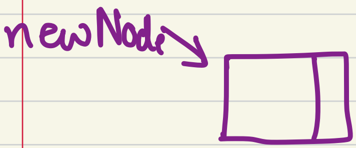
Then, we create an empty linked list

```
Node * head = NULL;        Node* tail = NULL;
```

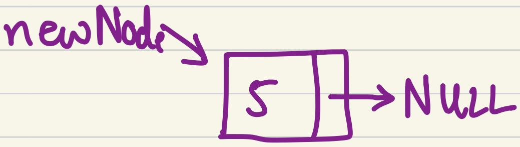
usually we don't have a tail pointer pointing to the last node.

Then, we dynamically create a node

```
Node* newNode = (Node*) malloc (sizeof(Node));
```

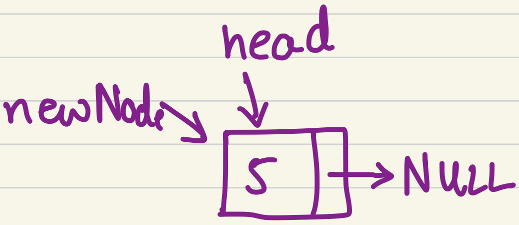


```
newNode -> data = 5;
newNode -> next = NULL;
```



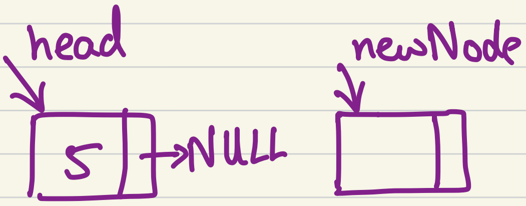
Then make the head pointer of the linked list point to the new node.

```
head = newNode;
```

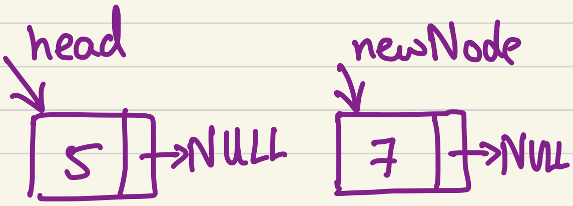


Then, we will want to create a new node dynamically.

```
newNode = (Node*) malloc (size of (Node));
```

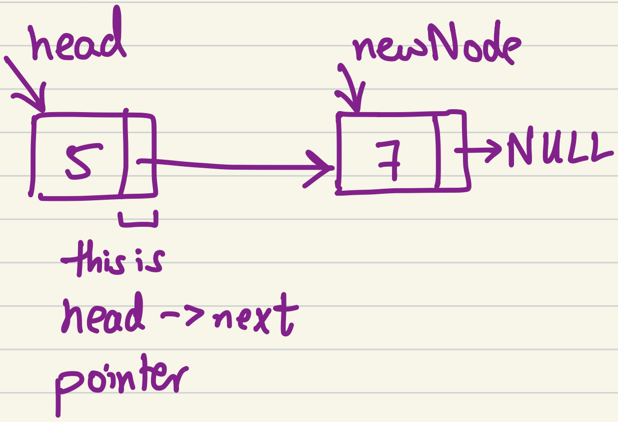


```
newNode -> data = 7;
newNode -> next = NULL;
```



Now, we need to dynamically link the two nodes together

```
head -> next = newNode;
```



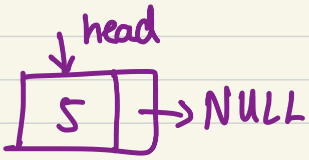
This is silly if we need to create many nodes in a linked list. We need a function for each operation.

Function 1: Create a function that allocates memory for a node, place a value inside data and return a pointer to this new node

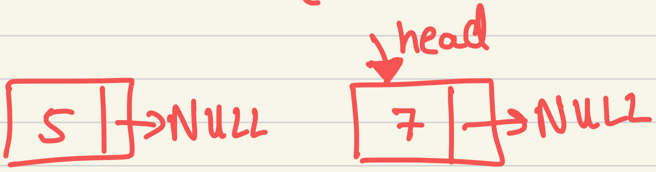
```
Node* createNode (int value){

    Node*   newNode = (Node*) malloc (sizeof(Node));

    if(newNode != NULL){
          newNode -> data = value;
          newNode -> next = NULL;
    }

    return newNode;
}
```

Sometimes it would fail to allocate memory due to lack of available memory on heap.

```
int main (){
    //Create a list with 1 node (in 1 line)

    Node *head = createNode(5);
```

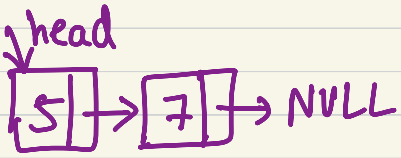


```
    //Create another node at the end of the list

WRONG → //head= createNode(7);
```

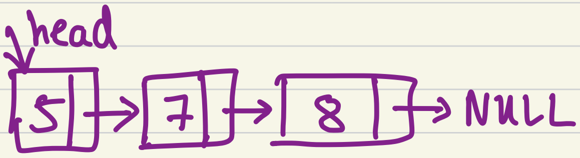


We lost access to our past node → we can never get back to it, we can never free it → Memory leak.

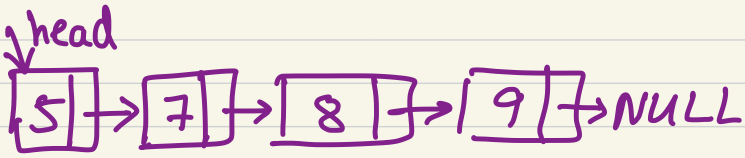
```
    head ->next = createNode(7);
```



```
    //Create another 2 nodes at the end of the list
    head -> next -> next = createNode(8);
```



```
    head -> next -> next -> next = createNode(9);
```

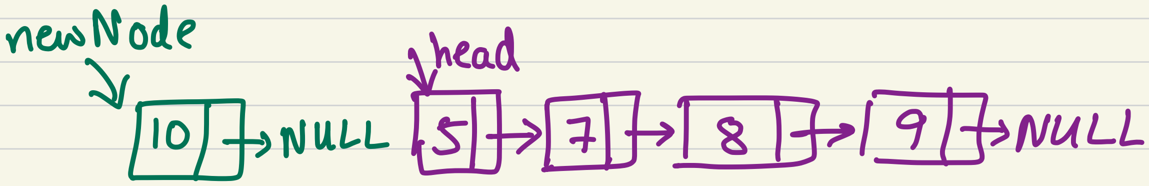


```
}
```

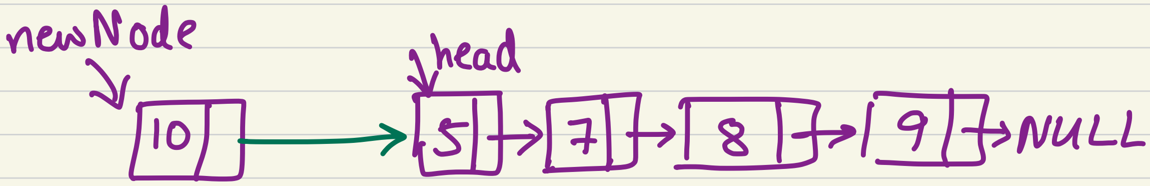
This gets silly if we will add multiple nodes!

Maybe it's better to add nodes before head at the beginning of the list.

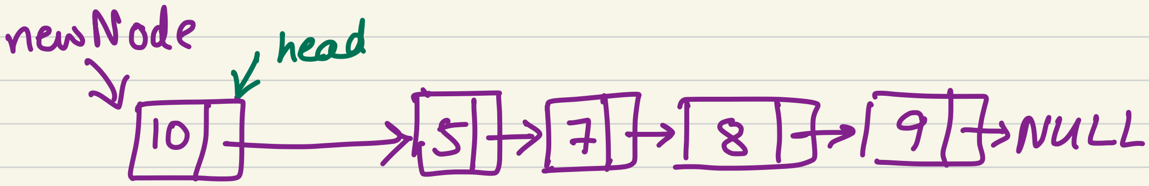
```
Node * newNode = createNode(10);
```



```
newNode -> next = head;
```



```
head = newNode;
```

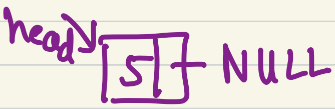


Let's write a function to insert a node at the head of the list. The function takes value in node to insert and pointer to head node. The function returns if we were successful in inserting the node.

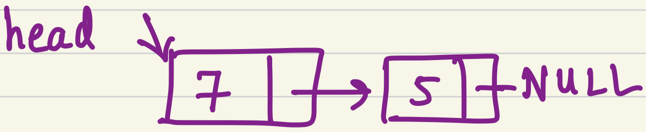
```
bool insertAtFront( Node* head , int value){

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

So now in main,

```
int main (){
        Node * head = NULL; → empty linked list

        insertAtFront(head ,5);
```



```
        insertAtFront(head, 7);
```

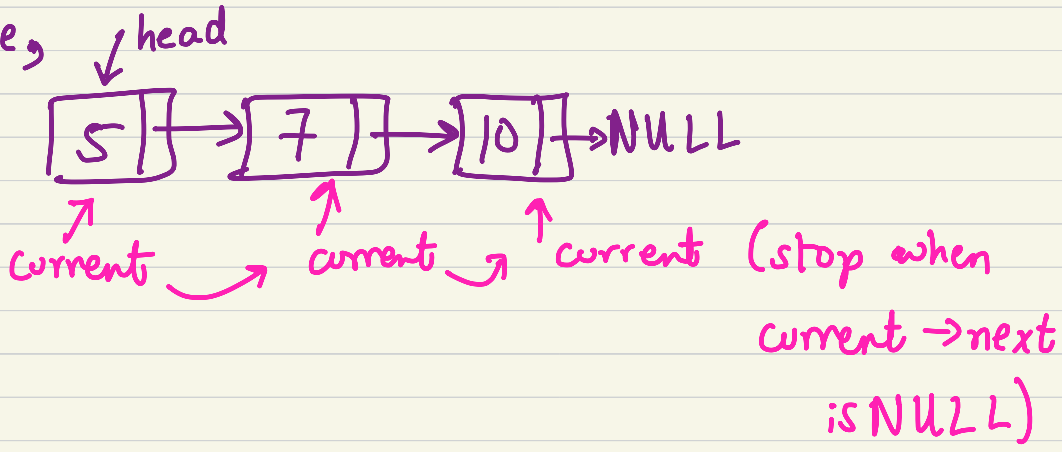


```
        return 0;
}
```

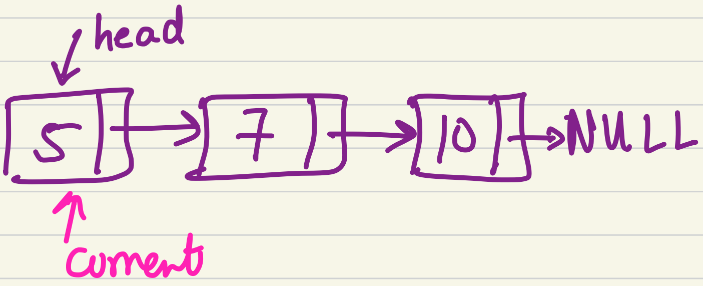
Let's have functions to print elements in a linked list

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

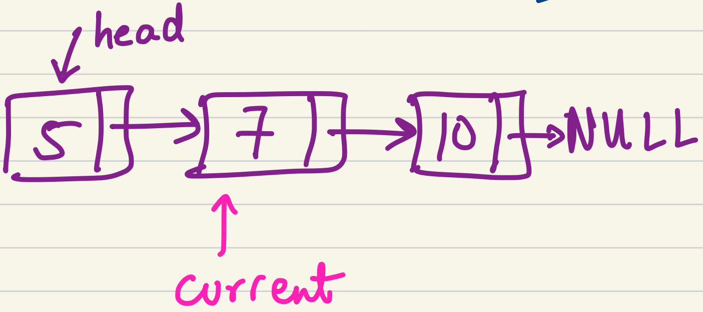
Example,



```
void printList(Node *head){

        Node * current = head;
```



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

                current = current -> next;
```



```
        }
}
```