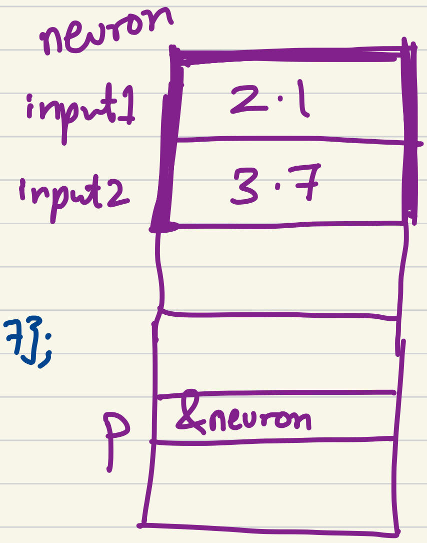
APS105 Lecture 29 Notes

Last time: We concluded discussion on recursion and we started discussing composite data types or user-defined structures using struct.

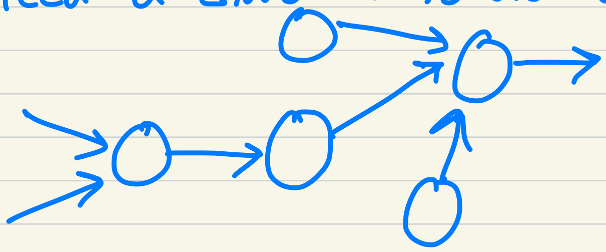
Today: We discuss how to form linked lists

Recap:

```
typedef struct Nstruct{
      double input1, input2;
} Neuron;

int main(){

      Neuron  neuron = {2.1, 3.7};
      neuron.input1 = 2.1;
      Neuron *p= &neuron;
    p ->input1 =  7.2  ;   //-> : arrow operator
    OR
   (*p). input1=  5.3  ;  //dereference a pointer

   //To dynamically allocate Date structure
    p= (Date*) malloc (sizeof(Date));
     ⋮
    free(p);
    return 0;
}
```



Now, we want to model neurons & synapses in the brain, we need a structure to do so.



⇒ too complicated to work with this complex structure!

Let's first work with <u>linear lists</u>

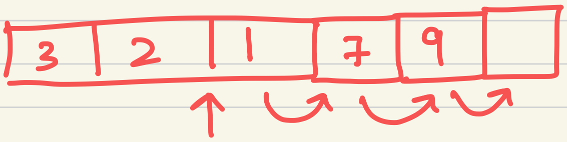
set of data stored in line

Previously, we sorted data in an array

e.g. `int N[100];`

Problems with arrays:

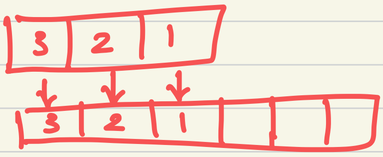
① Can't insert an element in the middle



Need to move all to the right to insert 7 in the middle

② Can't delete an item without having to shift elements to the left!

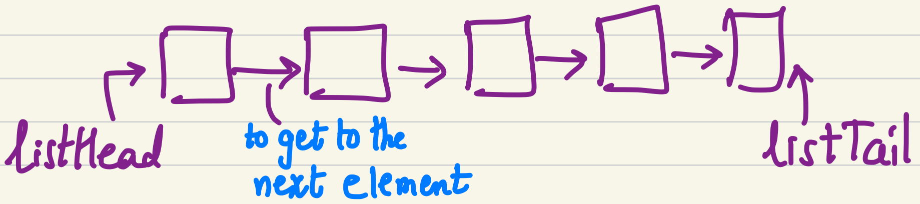
③ If I need more space, I will have to create a new array and copy elements.



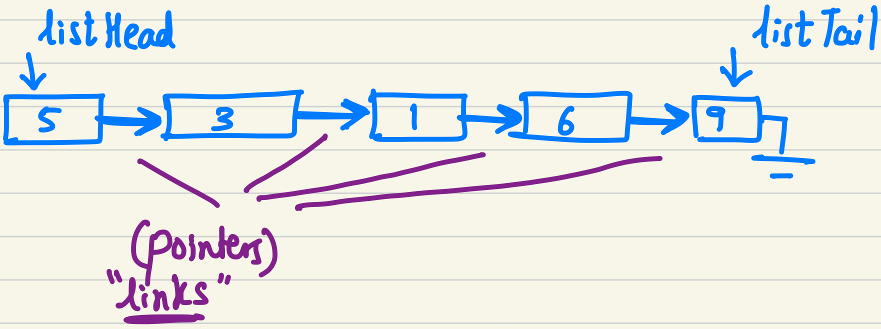
Instead we have linked lists

- Dynamically allocated data structures
- Can easily change / extend / delete / add elements

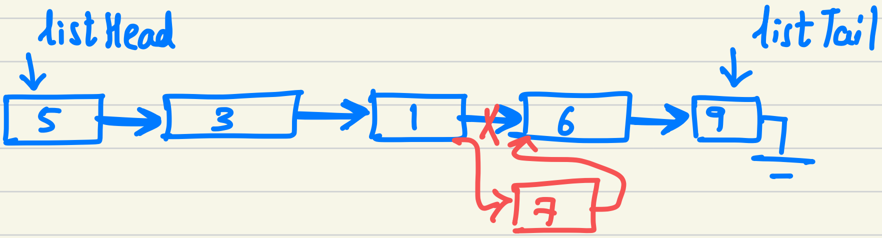
Divide up the array to have more flexibility



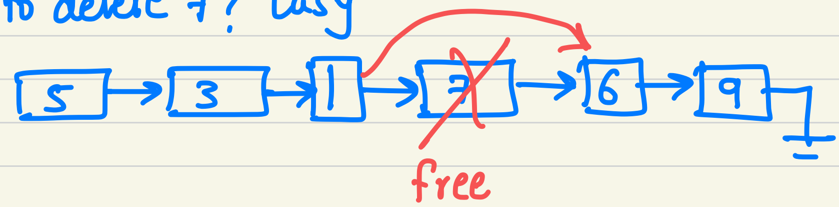
Visually:



How to insert 7? Easy



How to delete 7? Easy



How to implement this?

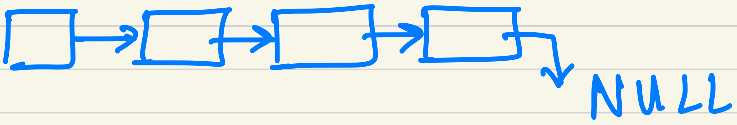
Each element is a user-defined data structure.

```
typedef struct node{
        int data;
        struct node *next;
} Node;
```

struct node → points to next node of type struct node

Note: It is acceptable to define a member of user-defined data structure in terms of itself.

How to identify last node?

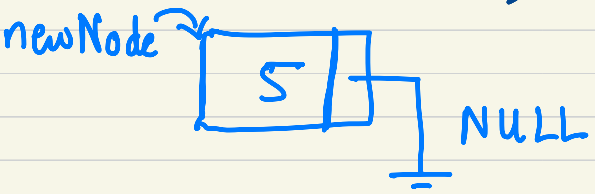


How to create an empty list?

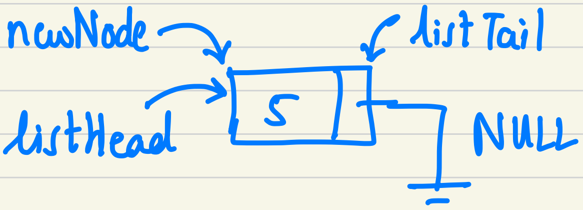
```
listHead = NULL;
```

Let's create an actual Linked List!

```
Node *listHead = NULL , *listTail = NULL;
Node *newNode = (Node*) malloc(sizeof(Node));
newNode -> data = 5;
new Node -> next = NULL;
```



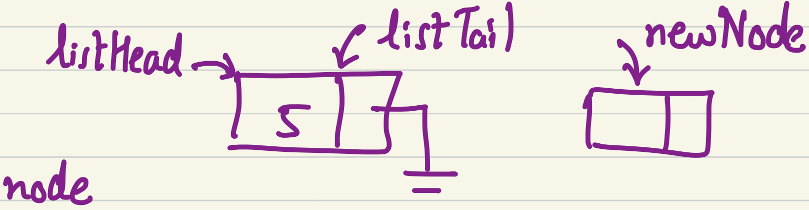
```
listHead = newNode;    listTail= newNode;
```



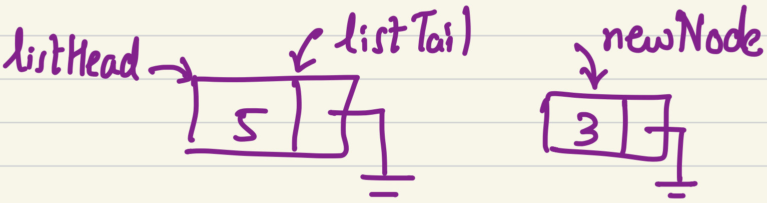
Let's add a node to the end of the list

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

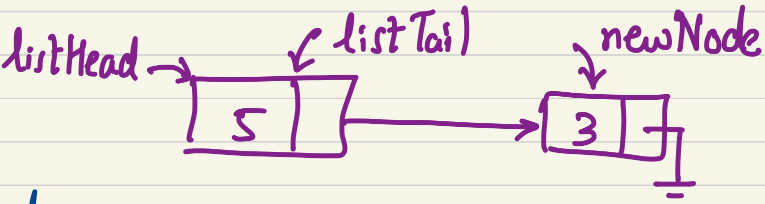
now New Node pointer is pointing to a newly allocated node



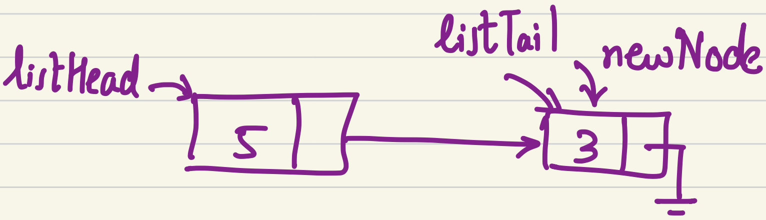
```
newNode -> data = 3;
newNode → next = NULL;
```



```
listTail ->next = newNode;
```



```
listTail = newNode;
```



General operatione we want to do on a linked list:

1) Add a node
2) Delete a node
3) Search list
4) free list