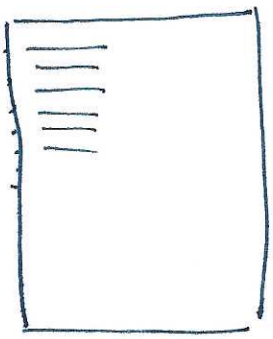
# CSE101 : The List ADT



All of Mark Twain's words (of length at least 6 characters)

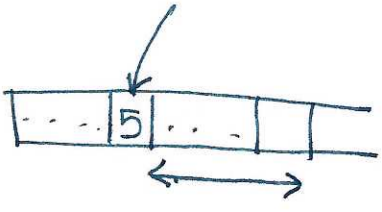
For any letter (say 'c'), what is the $k^{th}$ most frequent word?

Given a word, FIND if word is already present (seen)
UPDATE, frequency of the word
INSERT, if word has not been seen
DELETE, remove word

List ADT (Abstract Data Type)

For simplicity, list of ints.

Implement list as an array of ints




FIND, single for loop over the array
INSERT, insert at the end of the array
DELETE, shifting entries to the "left" using a for loop

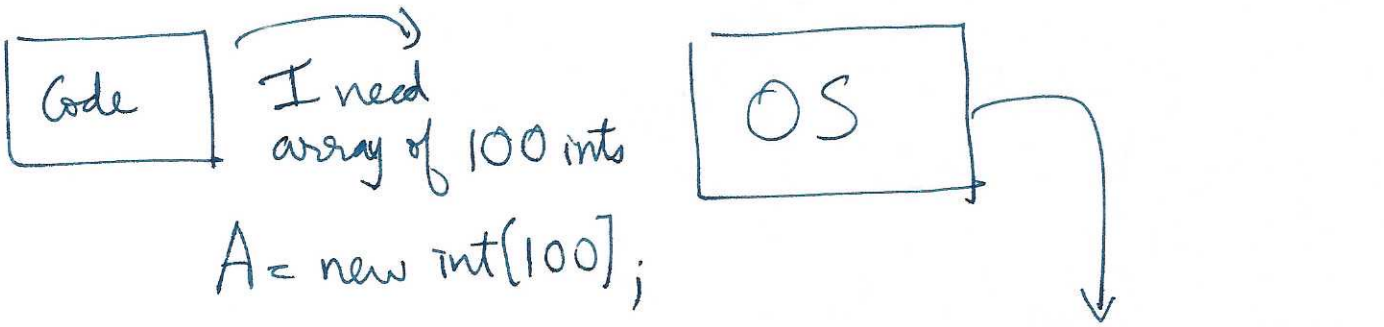
The problem:

We don't know a priori how long the list will be

How do we initialize the array size?

Double size / dynamic array / ...

Memory wasted.



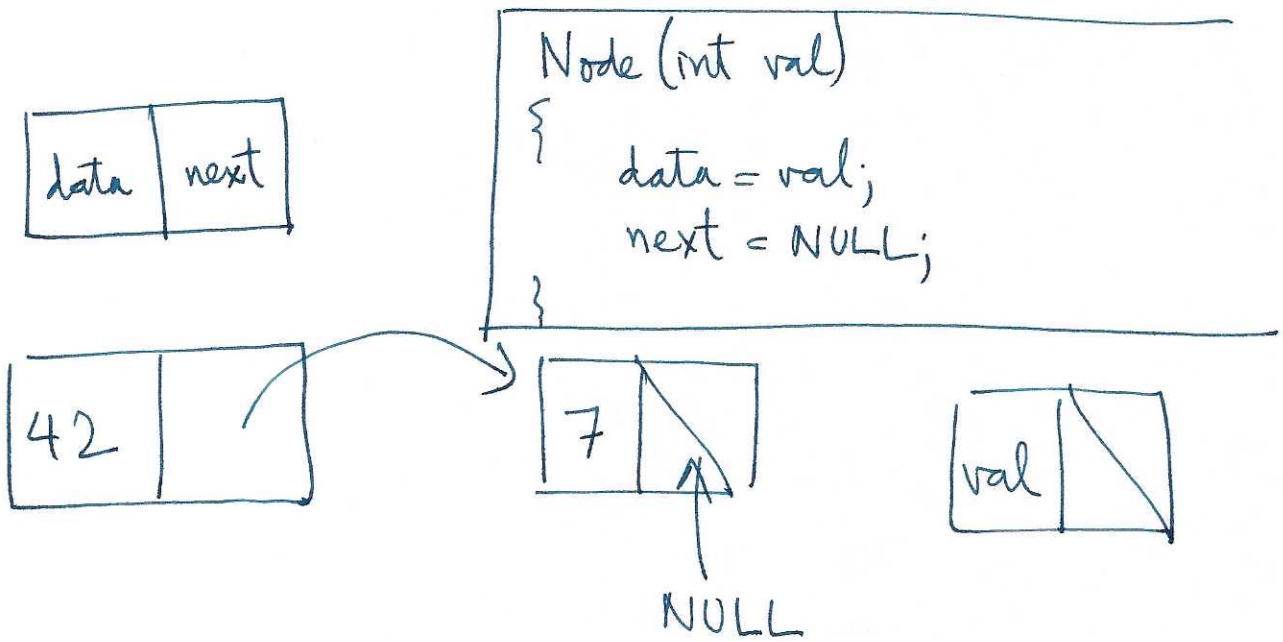

Linked list is a specific implementation of the List ADT

Benefit: It allocates "exactly" the right amount of memory.

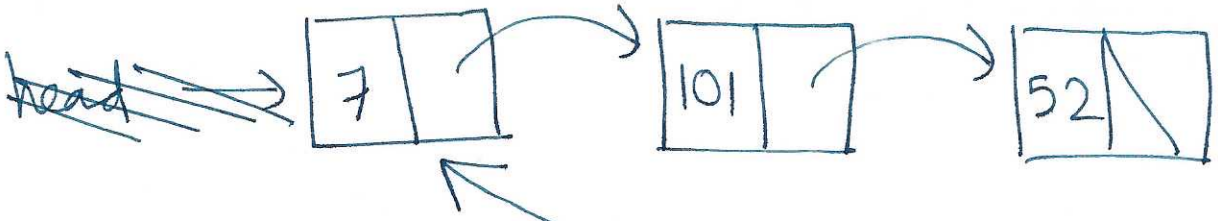
We don't need to know size a priori.

We don't need contiguous memory. beforehand

```
class Node
{
    int data; // in general, this can be an object
    Node *next; // pointer to the "next" node
}
```

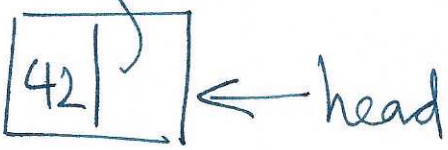



```
class LinkedList
{
    Node *head; // ptr to the first node of
                   the list
    ~~void~~ LinkedList(~~void~~)
    {
        head = NULL;
    }
```




insert(42)

Create a node with 42



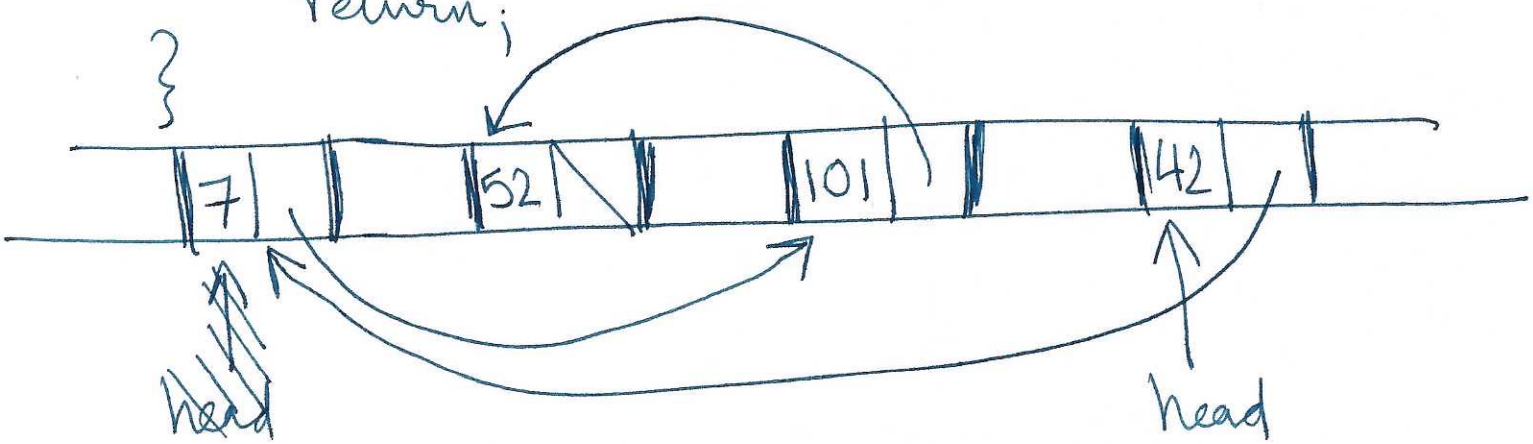

"link"

Make new node point to old head node

Make ~~no~~ head the new node

```
Node latest;
Node* latestptr;
```

```
void insert (int val)
{   Node latest; latest -> data = val;
    Node *latest = new Node(val); // creating memory for Node
    latest -> next = head; // linking
    head = latest; // reset head
    return;
}
```



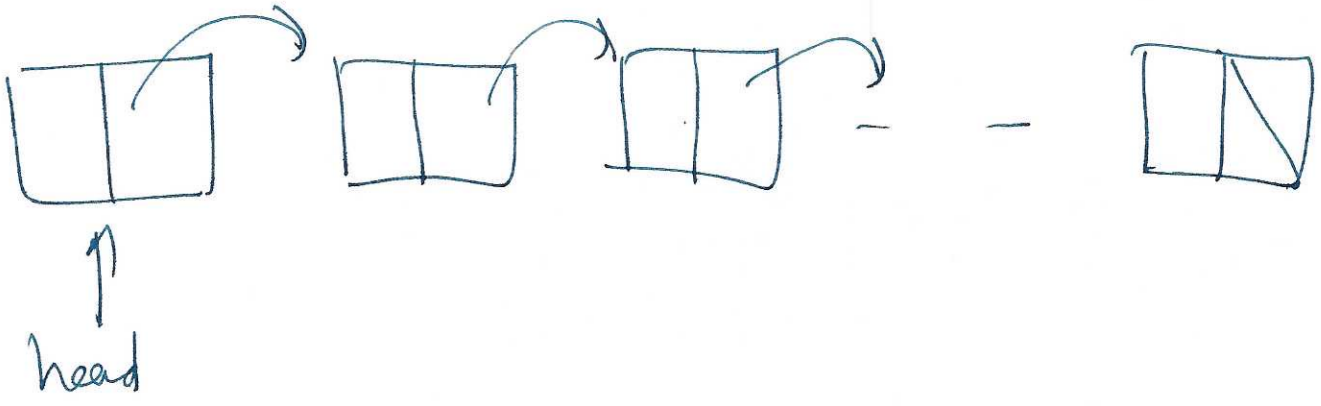
Node* find(int val)

↳ return a ptr to a node with val
return NULL if val is not present

find:

(1) Check if head -> data is val.

(2) If not, recursively find, starting from head -> next;




```
Node* find(int val)
{

    return find(val, head);

}
```

```
Node *find(int val, Node* start)
{
    if (start==NULL)
        return NULL;          // base case
    if (start->data==val)     // value is found!
        return start;
    return find(val, start->next);   // recursion
}
```

What?

Printing linked list in order

Recursion or using iteration

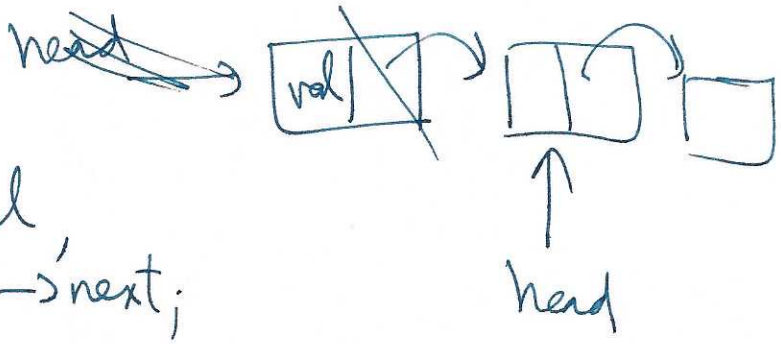
curr

```
Node* curr = head;
while (curr != NULL)
{
    print curr->data;
    curr = curr->next;
}
```

Node* delete(int val)

↳ return a node containing val (remove that node from list)
OR return NULL if val does not exist.

Think recursively




1) If head has val,
   head = head->next;
   return old head

2) Recursively delete, starting from head->next