Let's say we're implementing a linked list.

To represent list elements, we use a struct:

```
struct list_node {
    int data;          <- the value of the list element
    struct list_node *next;   <- pointer to the next
};                                element in the list
```

Here's a sample linked list containing values 3, 12, 6, 9:

For our example, let's start with an empty list...

```
int main(void){
    struct list_node *head = NULL;
    insert(head, 6);
    ...
}
```

pointer to NULL

When calling insert(), we copy head and 6 to pass by value

```
void insert(struct list_node *head, int val){
```

We want to:
- create a new node (malloc)
- update the old list head inside main

Inside insert(), we have:
- COPY of the address pointed by head (NULL)
- copy of 6

We can create the new node:

```
struct list_node *newnode = malloc(sizeof(struct list_node))
newnode -> data = val;
newnode -> next = head;
head = newnode;
```

— won't work! We are only changing a copy

How do we fix this? We can change 'head' inside the insert() function all day, but it's not going to modify the variable in main()!

Solution: let's pass a pointer to a pointer

(yikes!)

```
int main(void){
    struct list_node *head = NULL;
    insert(&head, 6);
    ...
}
void insert(struct list_node **head, int val){
    struct list_node *newnode = malloc(sizeof(struct list_node))
    newnode -> data = val;
    newnode -> next = *head;
    *head = newnode;
}
```

pointer to NULL

↳ Now we're passing in the address of 'head'

after insert: 'head' in main points at our new node, and its 'next' member points at NULL (the old head)

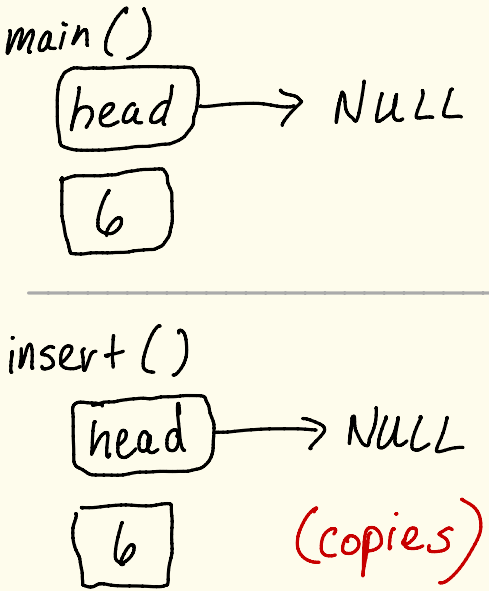
Inside insert(), we have:
- the address of the 'head' variable in main()
- copy of 6 (* NOT a pointer to NULL!)

We can create the new node:

next should point to the same thing as 'head' in main() (currently NULL)

go change what 'head' in main points at

Example 1 (single pointer)



Example 2 (double pointer)