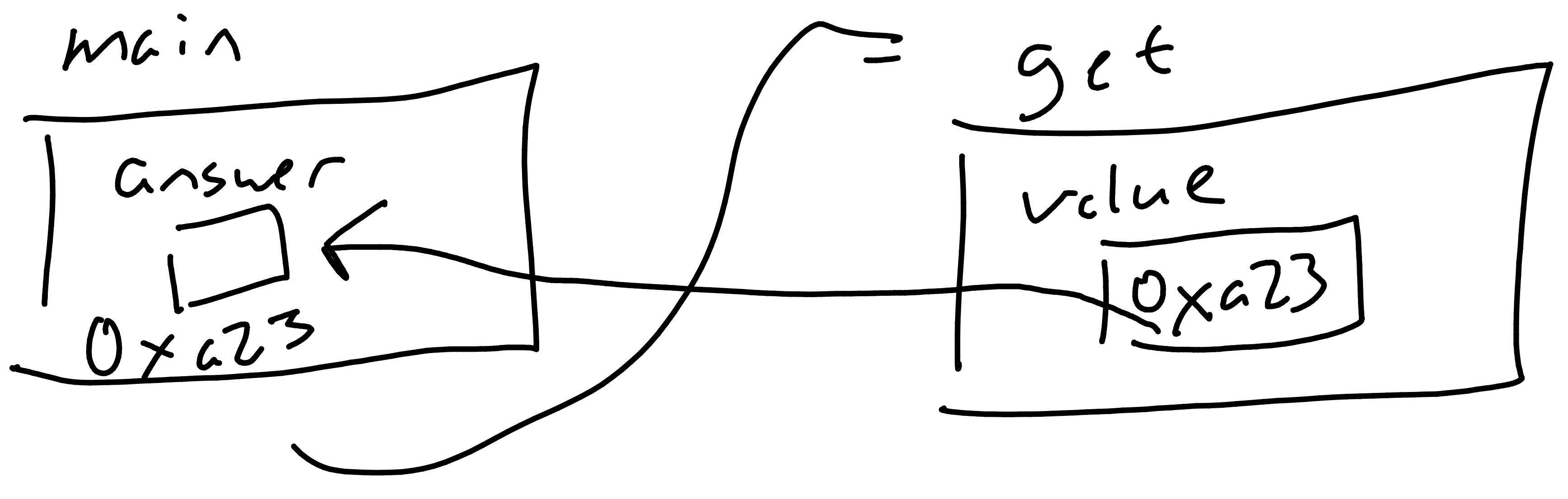
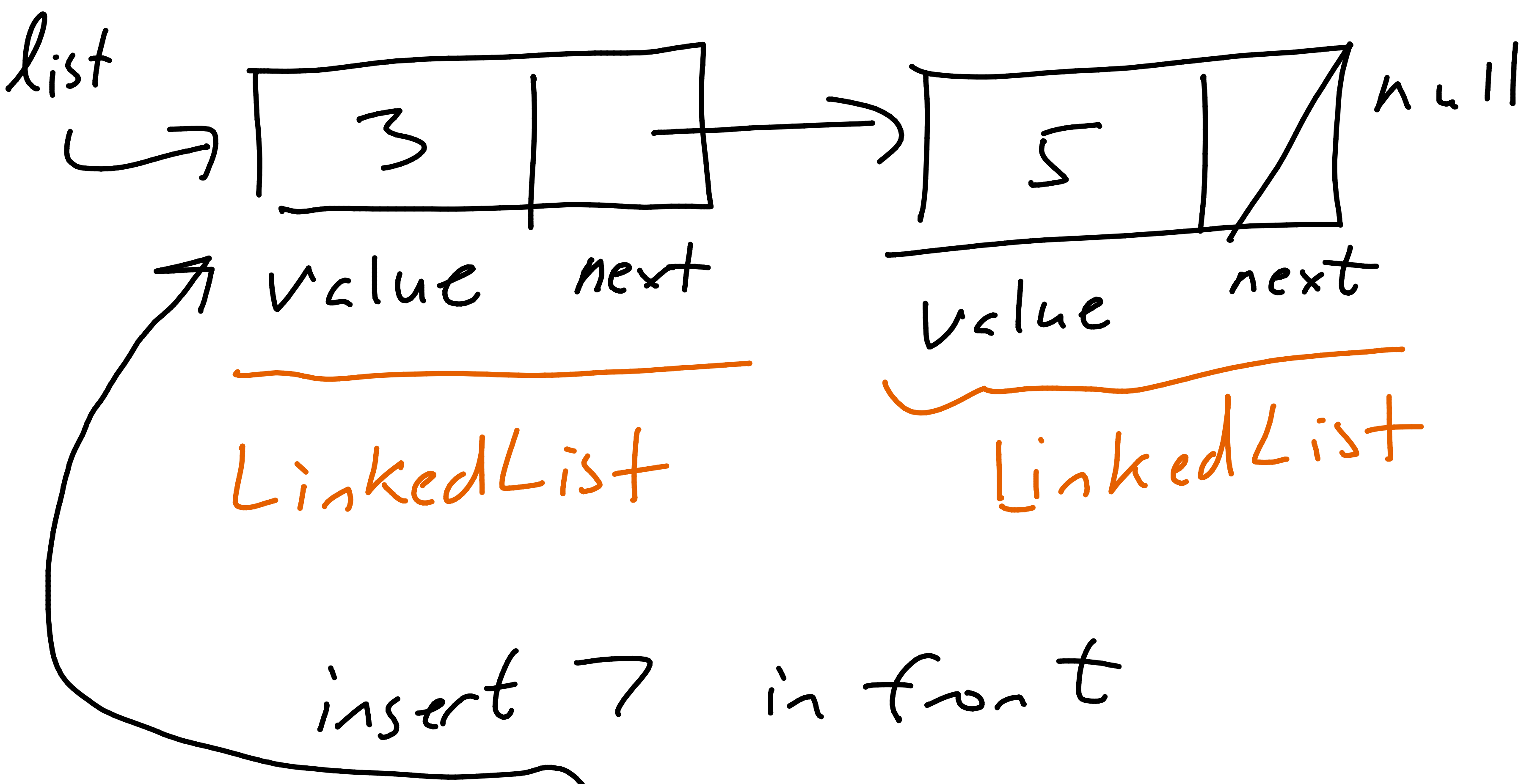
Today
- How do you do a linked list in C?
- What's the first step of the interpreter project?

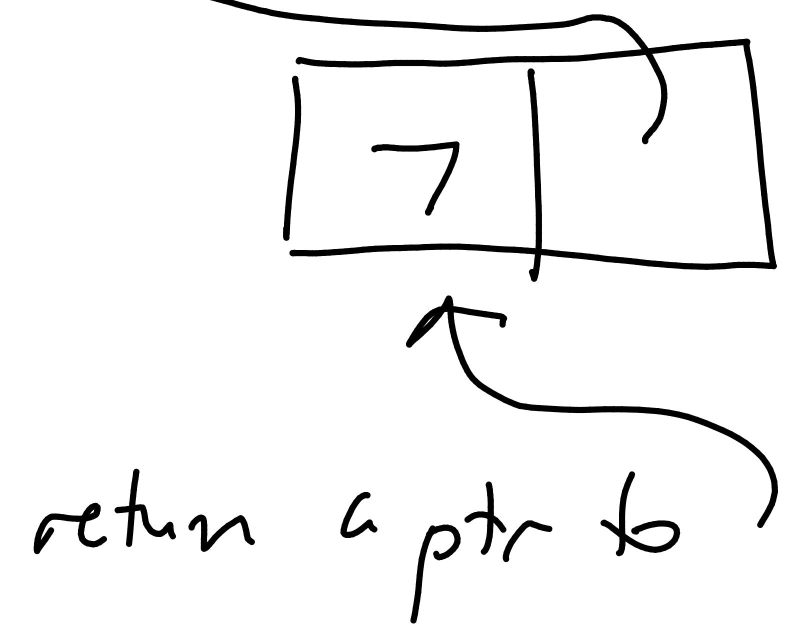
```
main() {
  int answer;
  int status = get(___, ___, &answer);
}
    :
int get(________)   {

    *value = ~____
    return 1
```



Our linked list will look like:

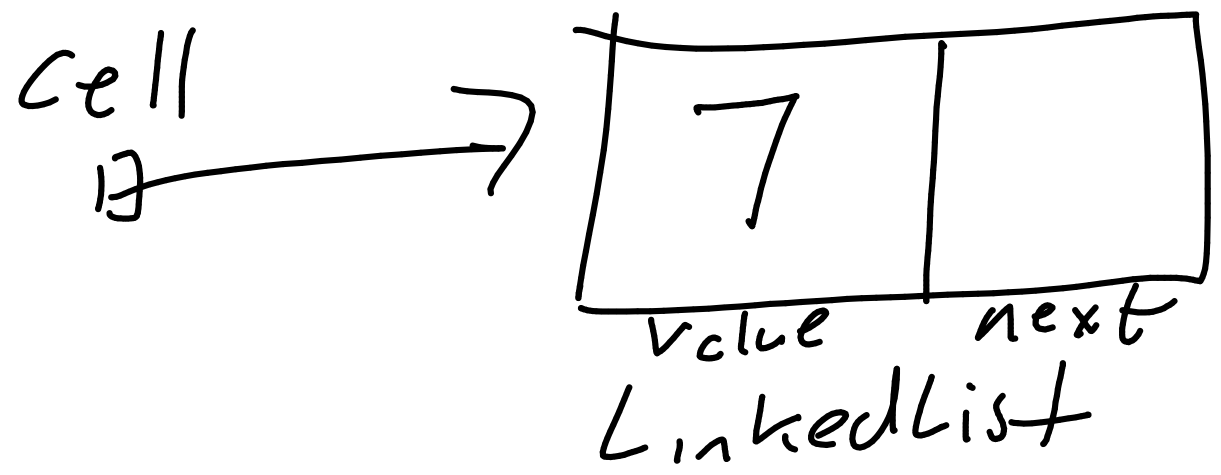


insert 7 in front

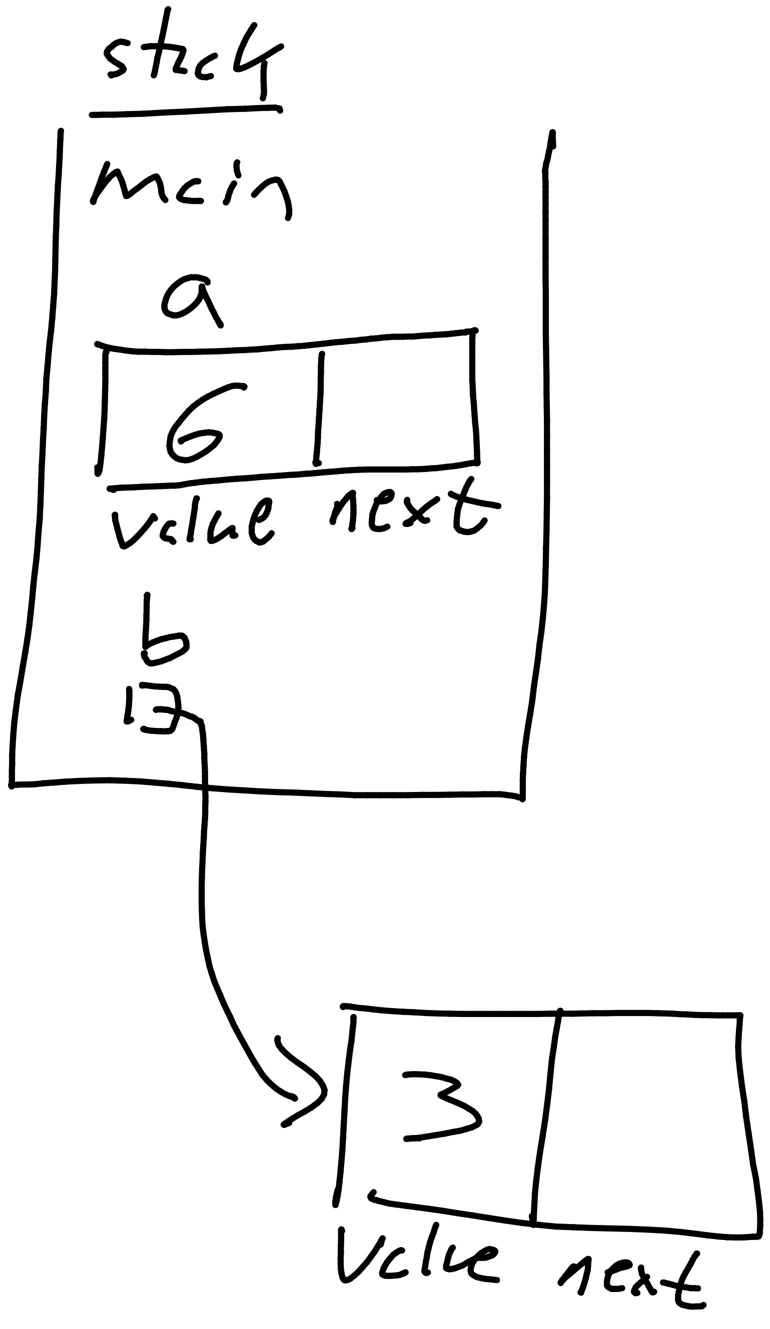


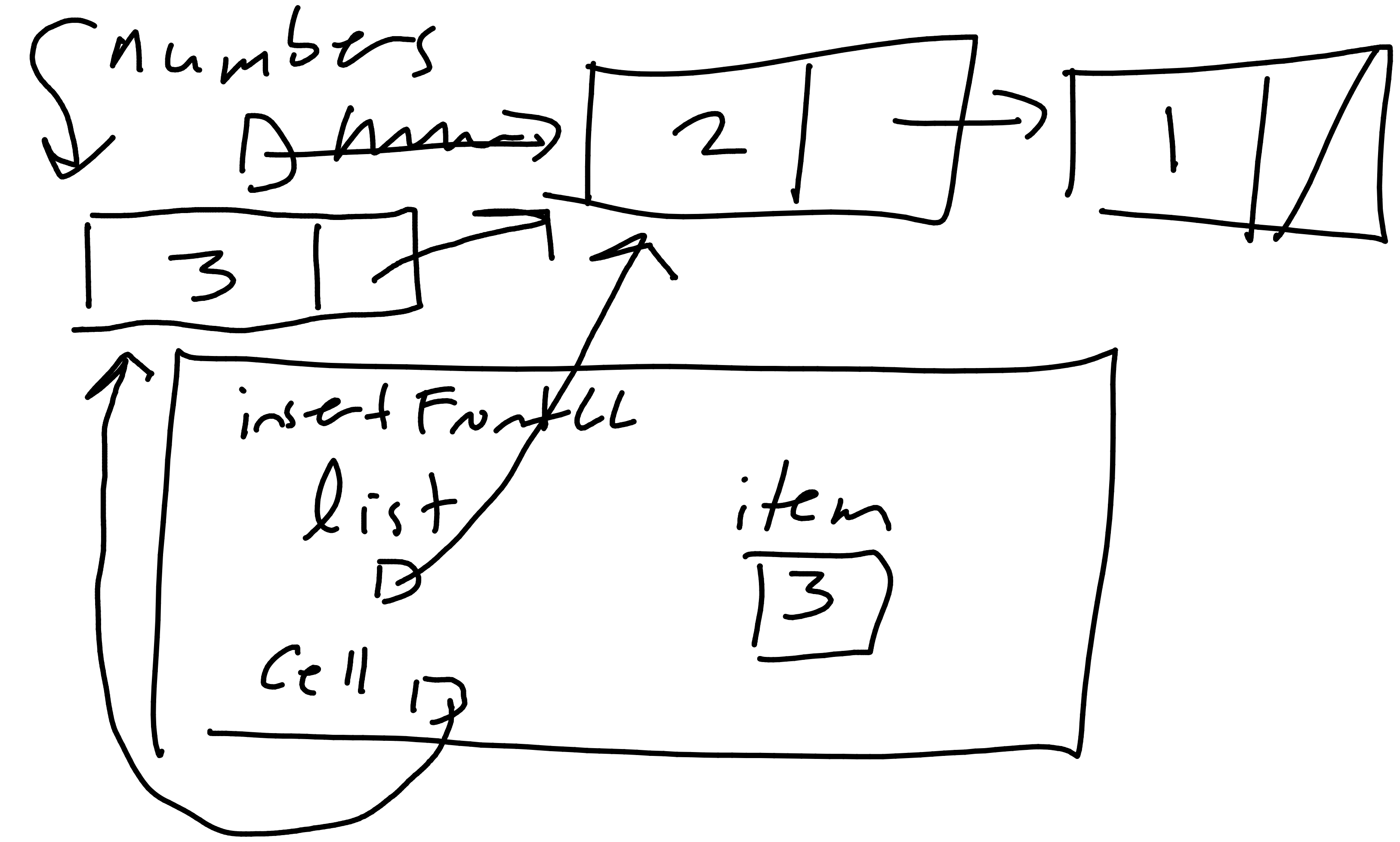
return a ptr to

again!

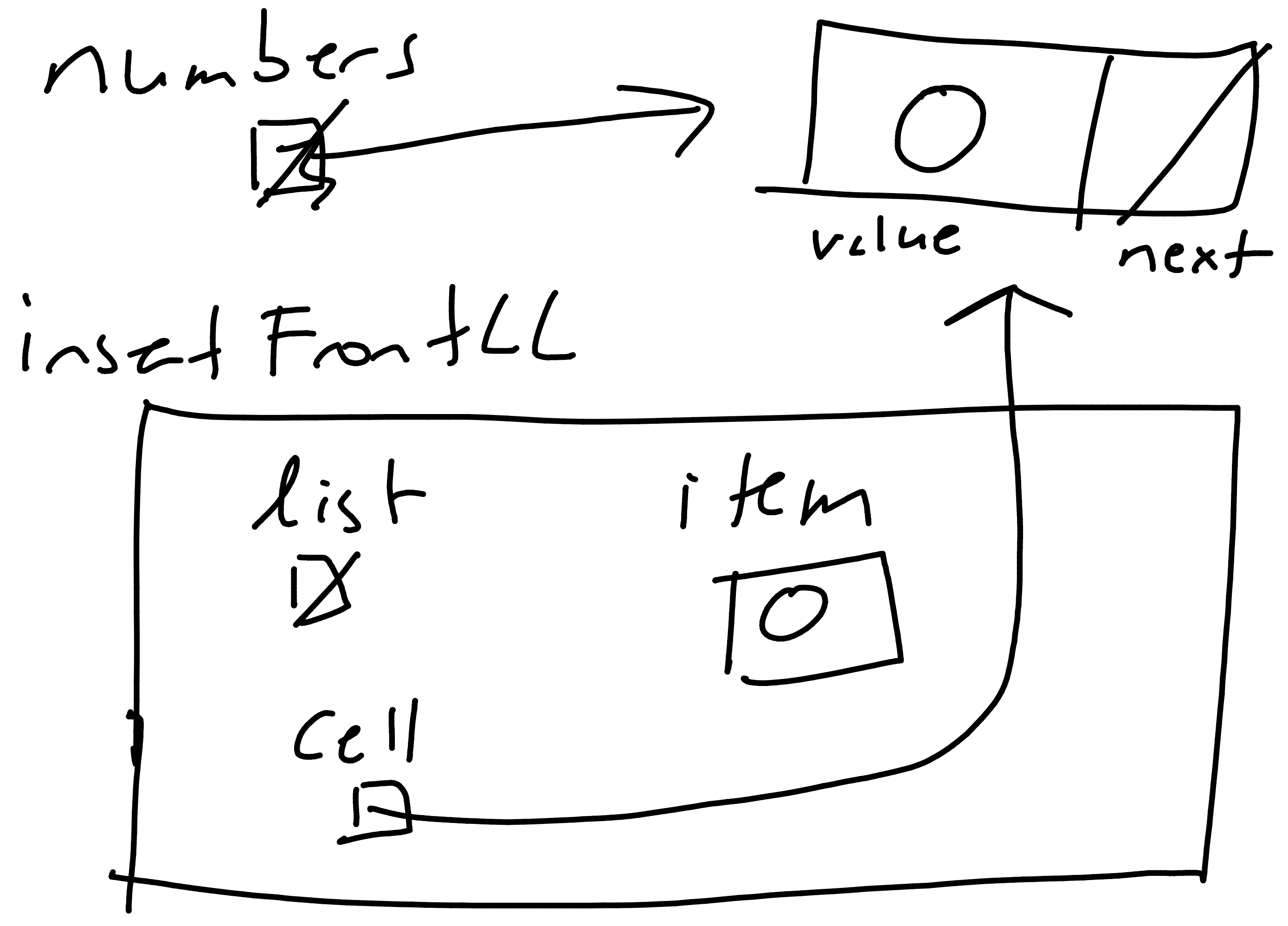


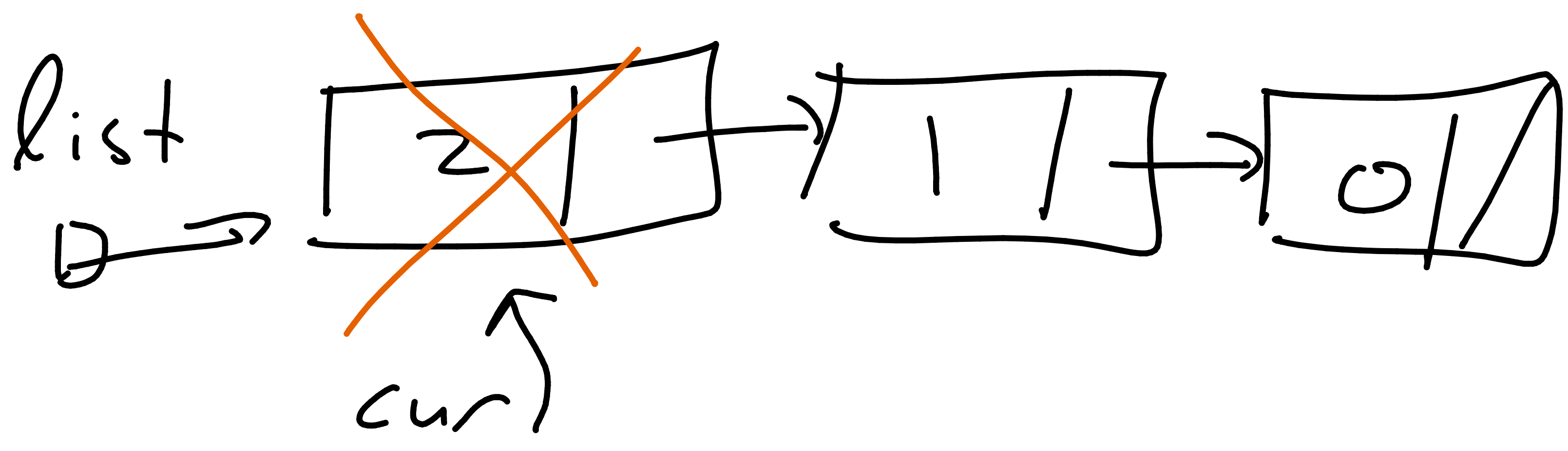
```
main() {

 LinkedList a;
 a.value=6;
 LinkedList *b;
 b=malloc .....
 ~~b.value=3;~~
 (*b).value=3;
  OR
  b->value=3;
```





Initially





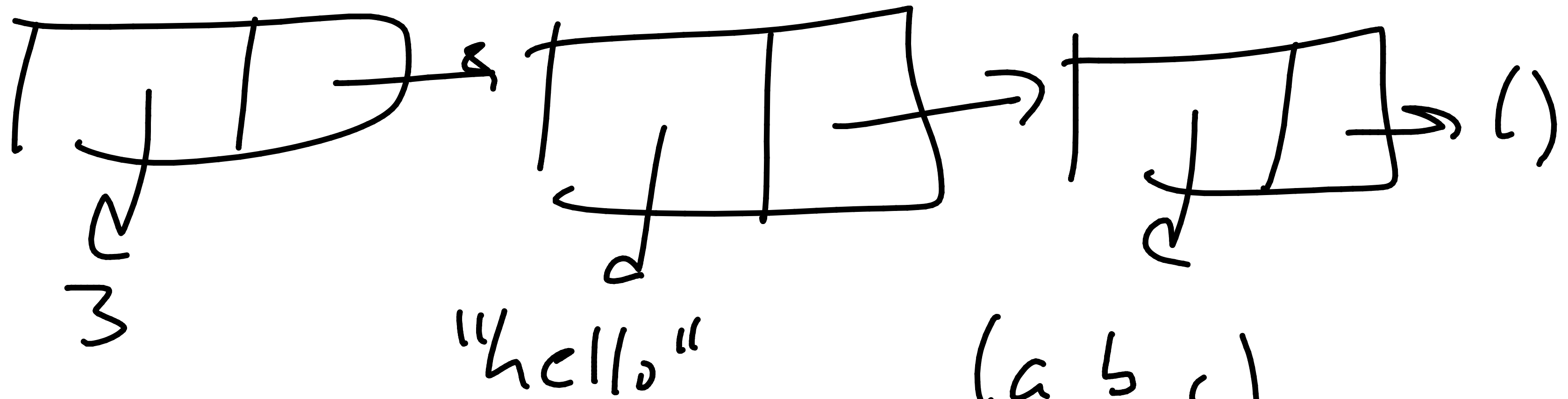
free(list)

↳ frees whatever the pointer directly points to

1st step of interp project: build a linked list
- partner assignment

Linked list of .. what?

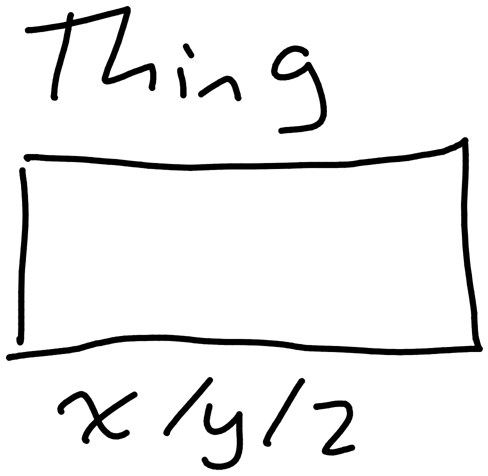
Scheme allows lists of mixed types



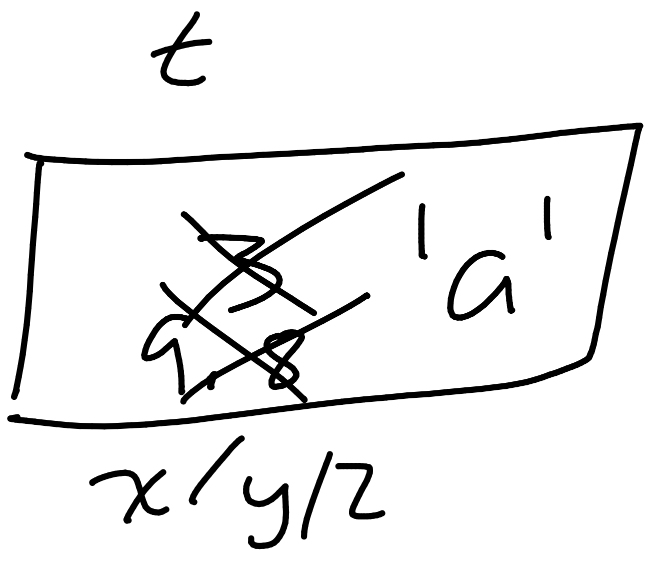
In C, if you want something to represent different types, one approach is to use a union.

→ looks just like a struct, but all values share the same memory

```
union Thing {
    int x;
    double y;
    char z;
}
```



```
Thing t;
t.x = 3;
t.y = 9.8
t.z = 'a'
printf("%i\n", t.x);
```

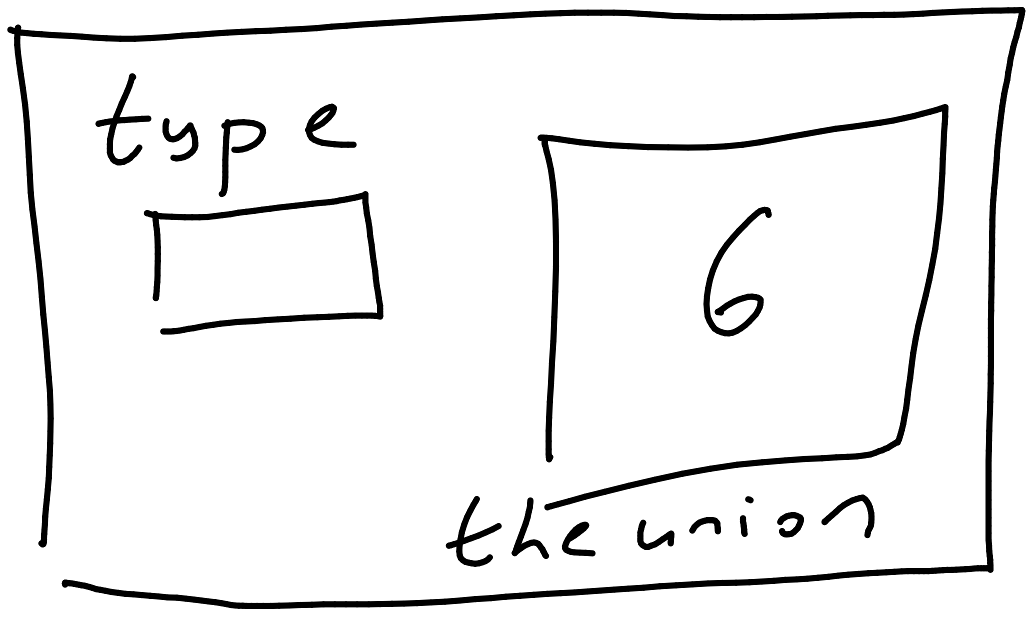


↓

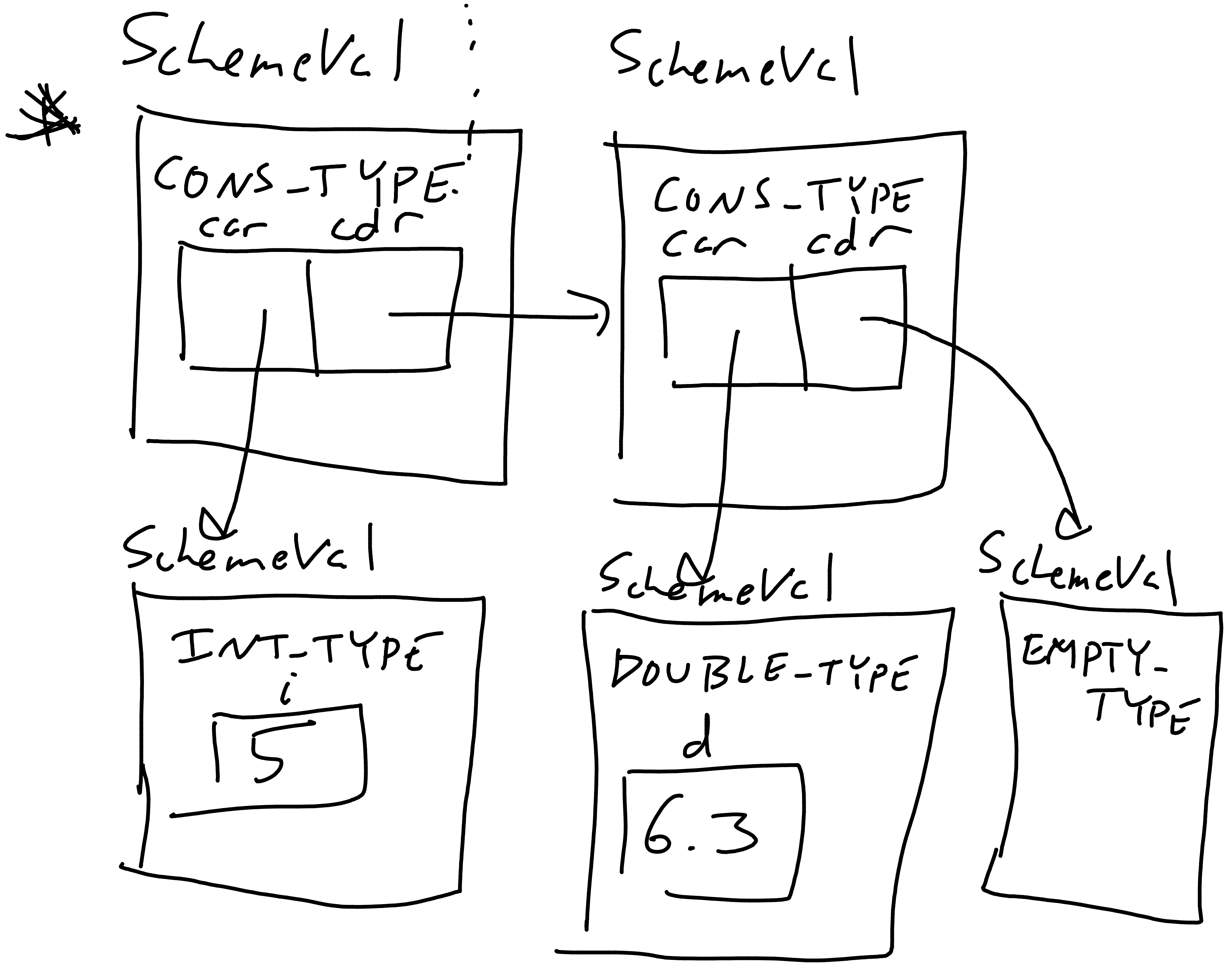
I'll get some number, based on reading the bits of whatever is in there

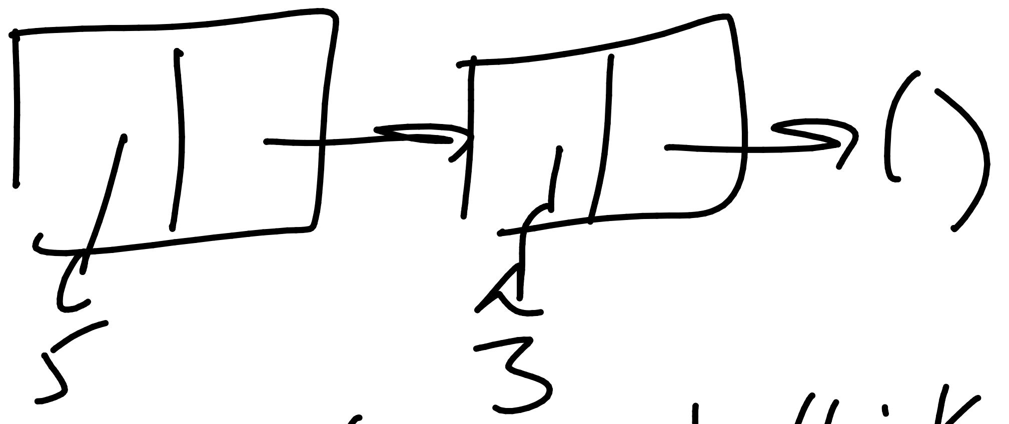
unions are great for multiple types, but you have to remember what you put in

SchemeVal ← struct



```
SchemeVal s;
s.type = INT_TYPE;
s.i = 6;
  :
  :
if (s.type == INT_TYPE) {
    printf("%i\n", s.i);
} else if ......
```





pseudocode (I didn't think about *s, etc)

```
list = malloc(sizeof(SchemeVal))
list -> type = CONS_TYPE
list -> car = malloc(sizeof(SchemeVal))
 list -> car -> type = INT_TYPE
list -> car -> i = 5
list -> cdr = malloc(sizeof(SchemeVal)
            .
            .
            .
            .
            .
```