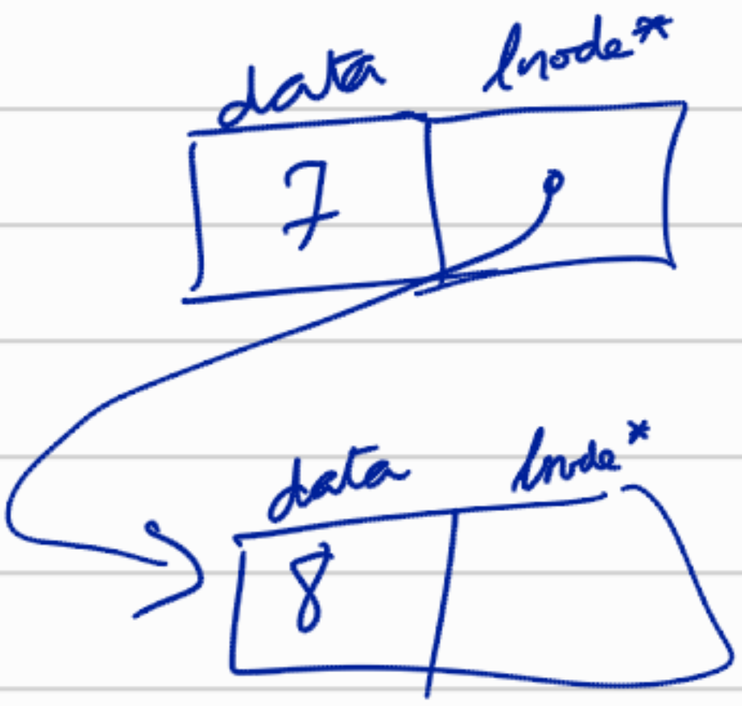
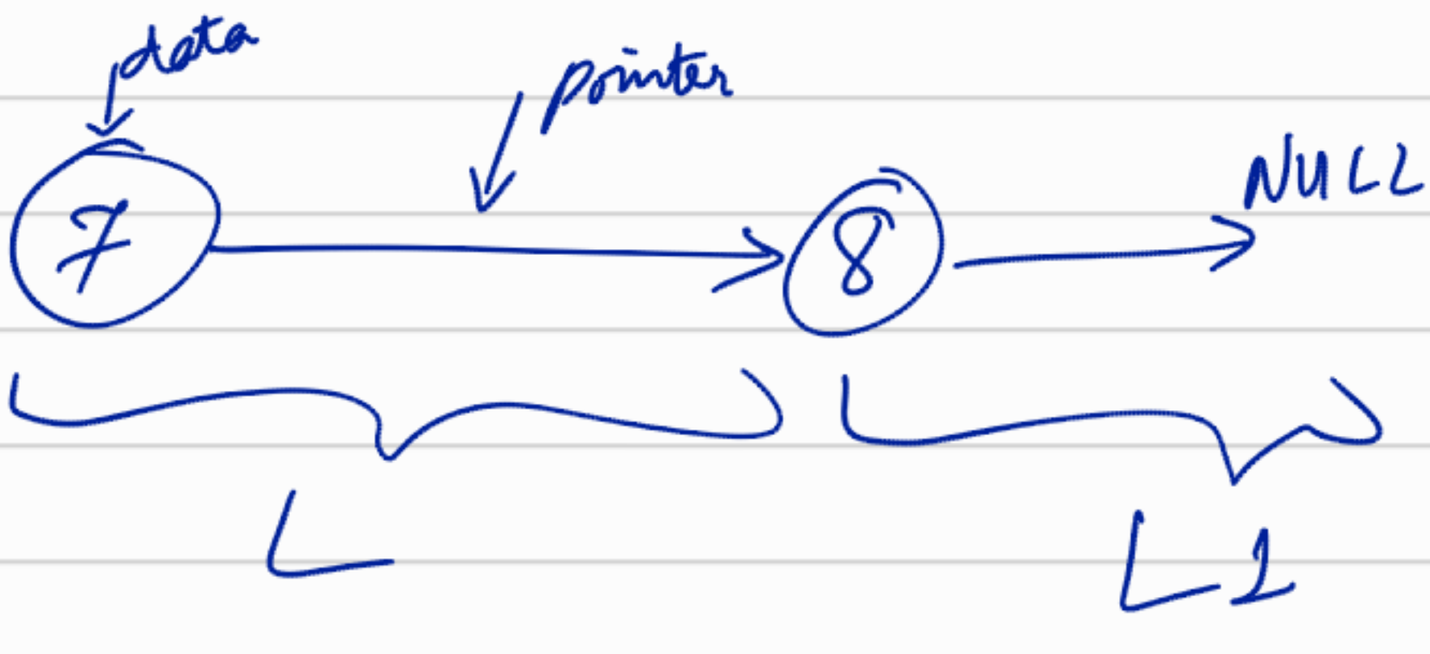
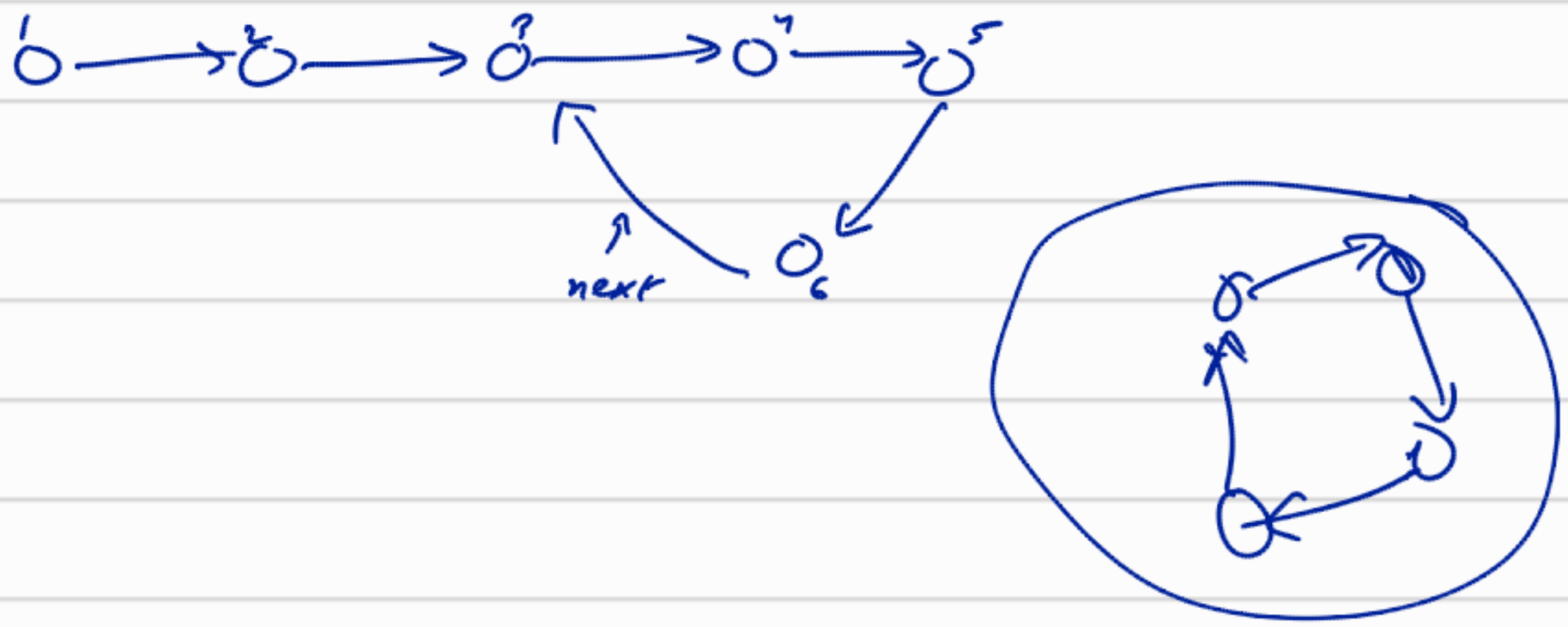
Linked lists : (LL)



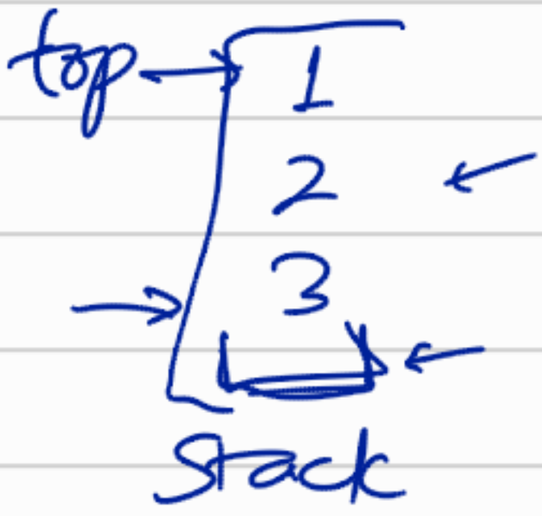

```
typedef struct list_node lnode;

struct list_node {
    int data;
    lnode * next;
};
```



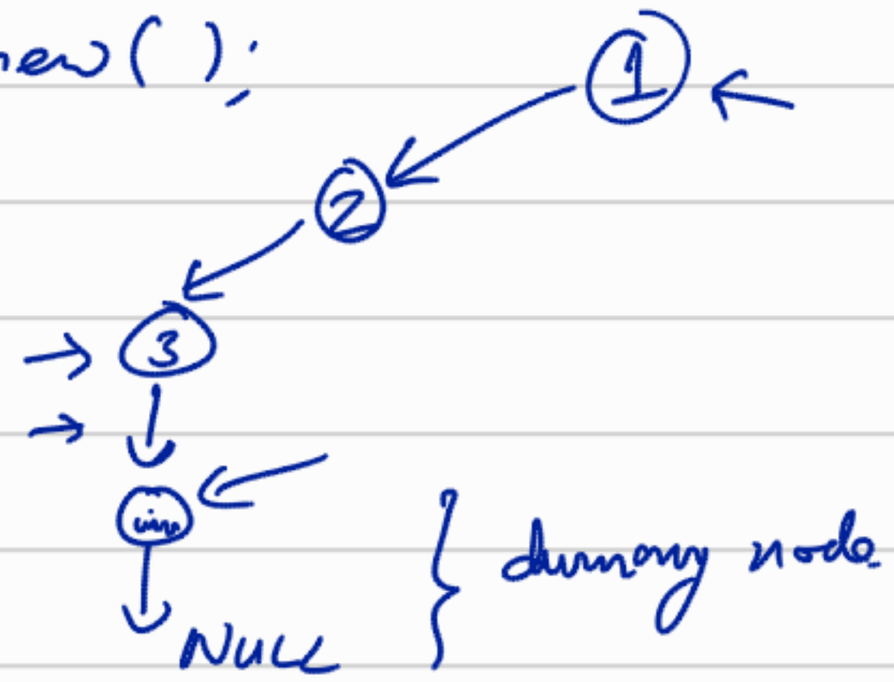
What can we create using the unit of LL.



Creating Stack using linked lists :-




```
stack_t S = stack_new( );
push(1, S)
push(2, S)
push(1, S)
```



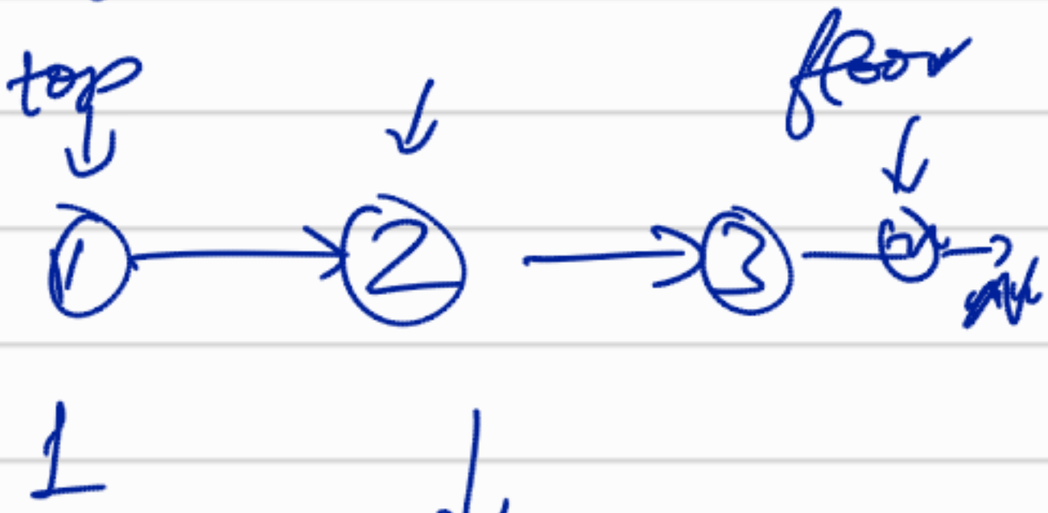
Stack :

```
lnode * top ;
lnode * floor ;
```

Create a stack :

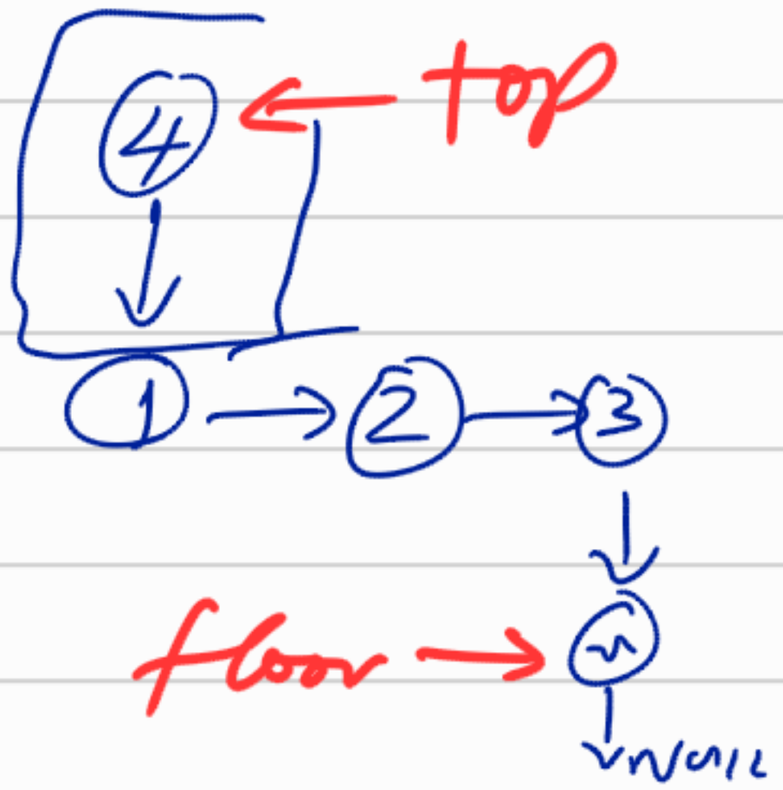
alloc a stack ;
alloc a dummy node :
fix the values of top = floor = dummy .
return the stack.

stack - pop :-

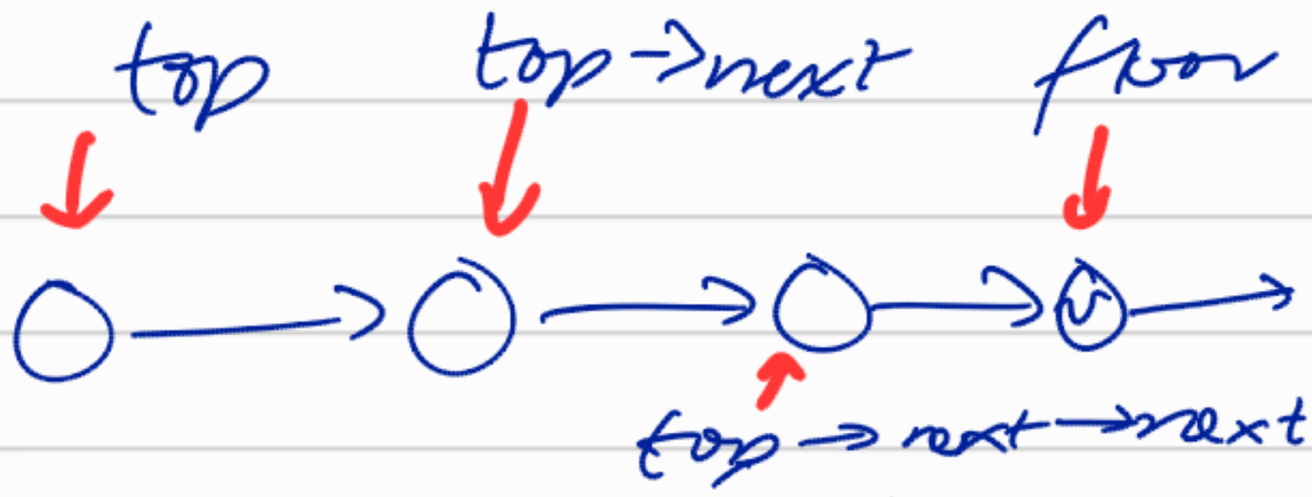



```
int x = S -> top -> data = 1
S -> top = S -> top -> next
return x ;
```

```
stack_push ( S, 4 ) {
    Lnode* nt = alloc(lnode);
    nt → data = 4
    nt → next = S → top ;
    S → top = nt ;
}
```

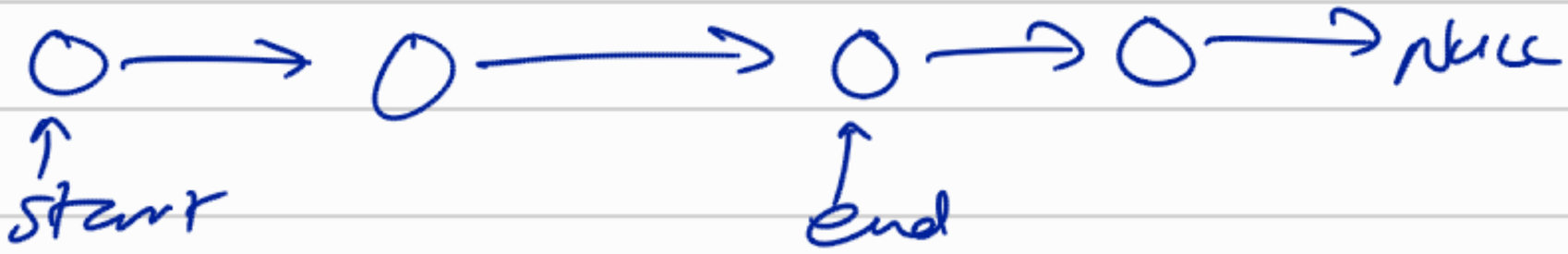


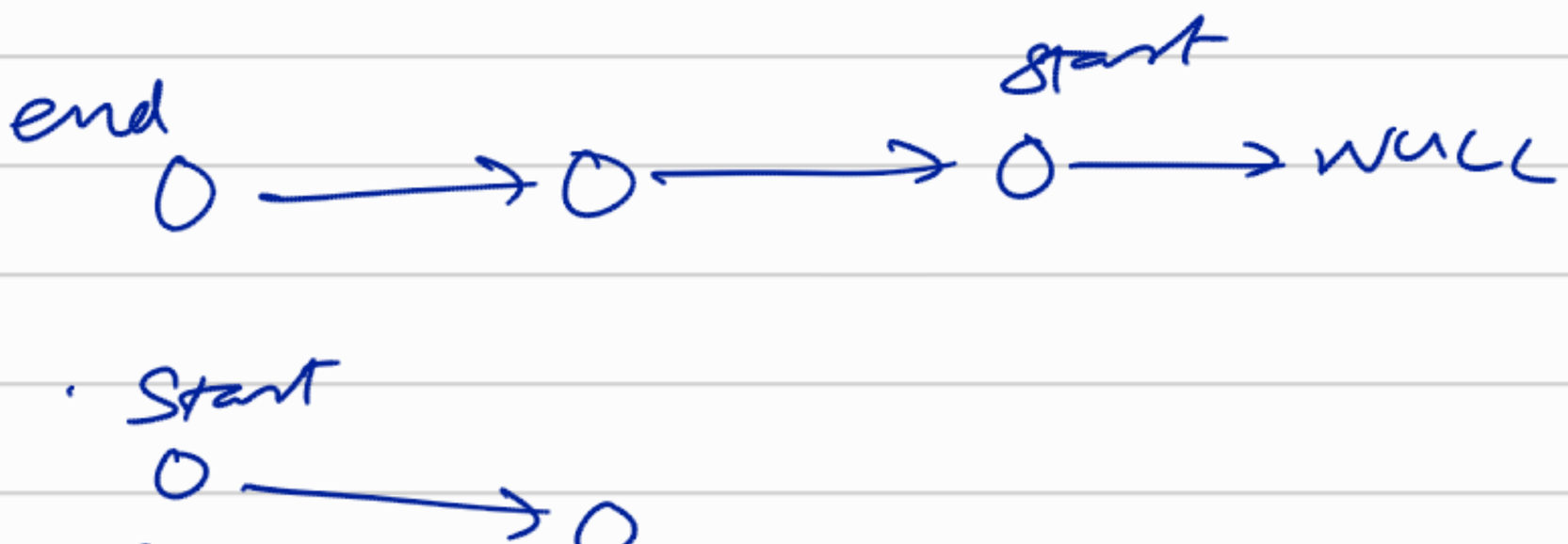
Assumption:

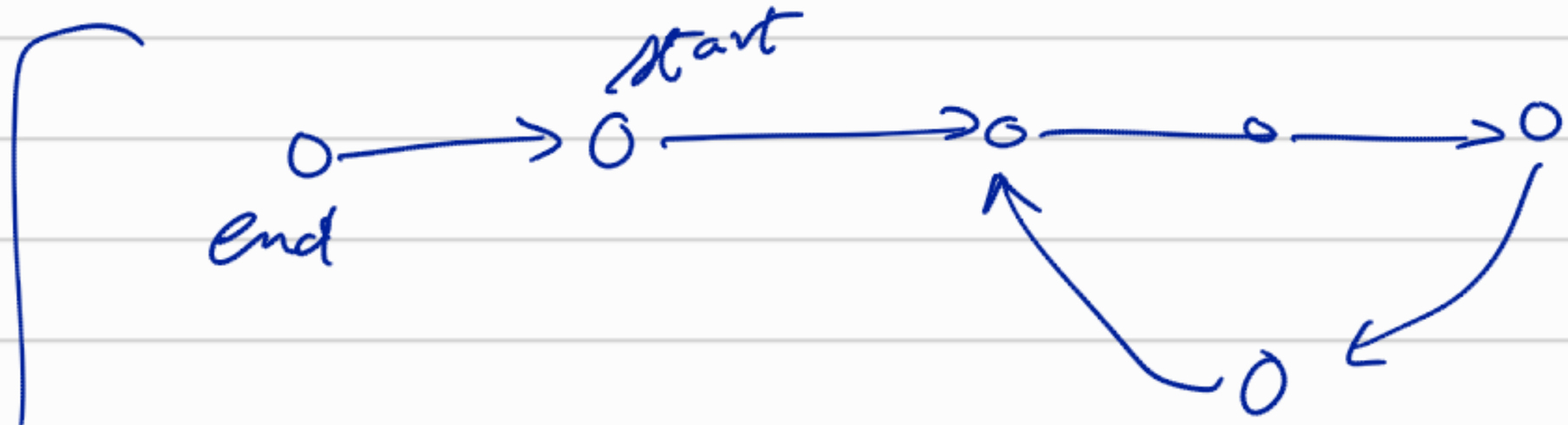


① the LL must be a segment.
meaning that there should be a way to reach the floor points from top using the next pointers.

```
bool is-segment (lnode* start, lnode* end)
    //ensure start != NULL;
{
    if (start == NULL) return false;
    if (start == end) return true;
    return is-segment (start -> next, end );
}
```





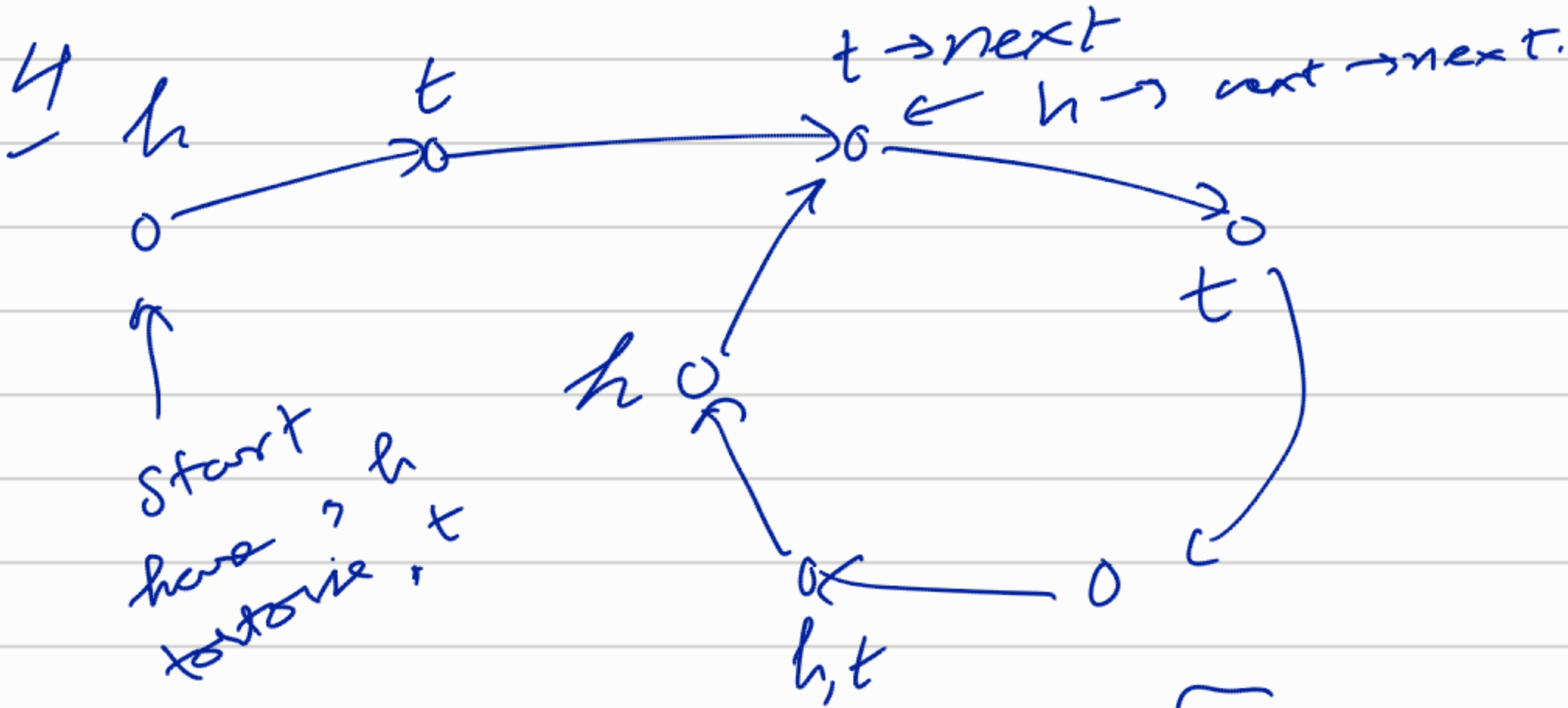


Requirements for a stack:

① L.L. must be a segment.

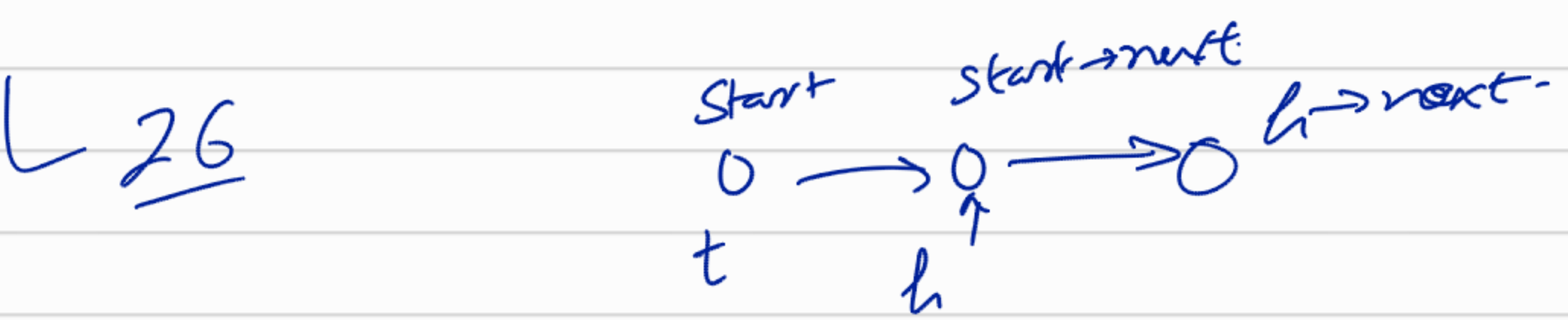
② L.L. must not have any cycles.

```
bool is_stack ( stack * S ){
    return S != NULL
        && is_acyclic ( S -> top )
        && is_segment ( S -> top, S -> floor );
}
```




```
while (h != t){
    t = (t -> next);
    h = h -> next -> next;
}
```

```
typedef struct list_node lnode;

struct list_node {
    int data;
    lnode * next;
};
```



Comparing Linked lists & arrays :-

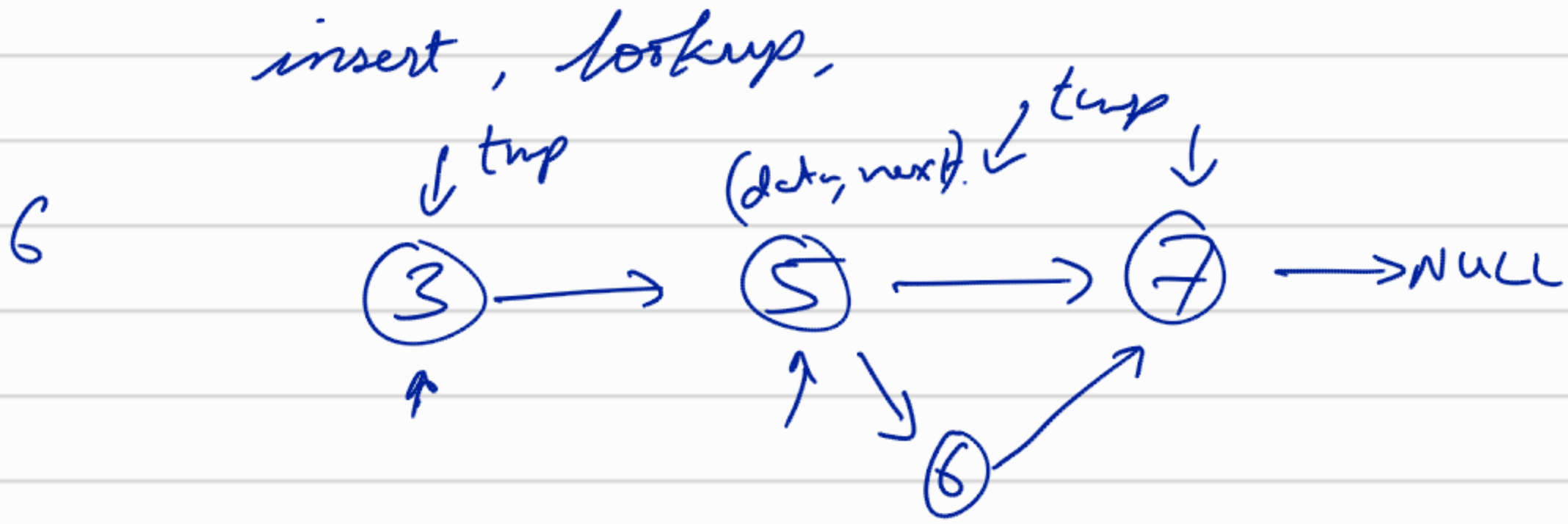
Linked lists	arrays
1) can have any desired size	(the size needs to be declared before allocating.
2) inserting elements is constant time (O(1)).	inserting depends on space.
3) looking up elements O(n) in worst case	look up stuff in O(log n) time if array is sorted.

Hashing :- O(1)

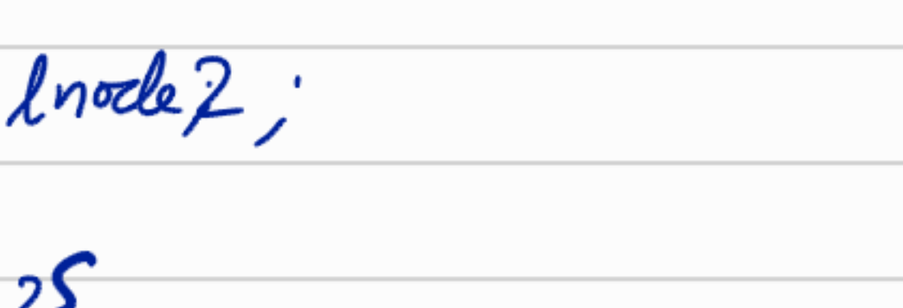
```
typedef struct list_node lnode;

struct list_node {
    int data;
    lnode * next;
};
```

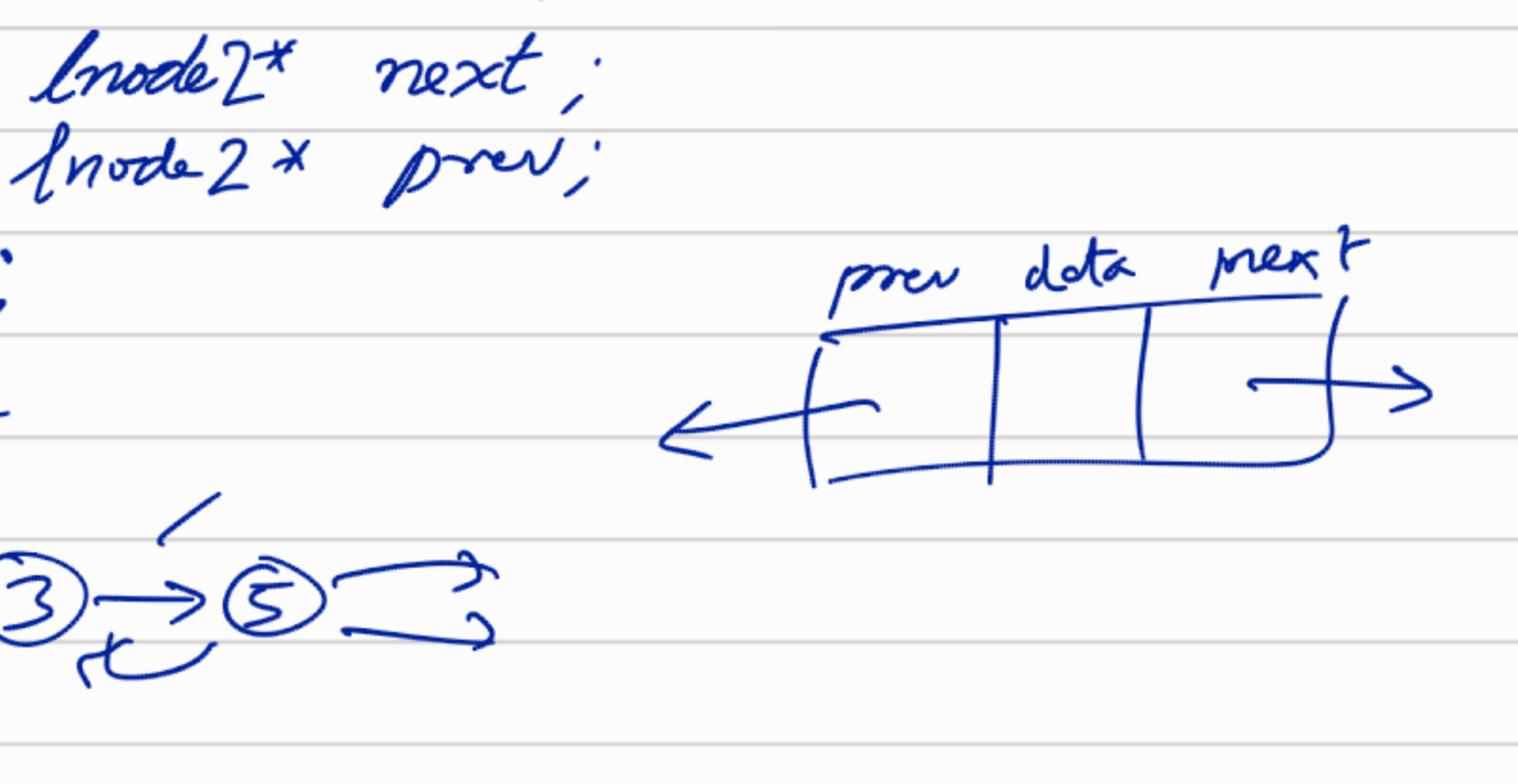
```
struct linked_list llist {
    lnode * front;
}
```

insert, lookup,



```
typedef struct list_node lnode2;

struct list_node2{
    int data;
    lnode2* next;
    lnode2* prev;
};
```





Co doesn't have generic pointers :-
generic pointers allow us to write codes which do not declared the type explicitly before hand.

Co → (C1/C).

filename . C1 — coin. doesn't work.
CC0 compiler works.

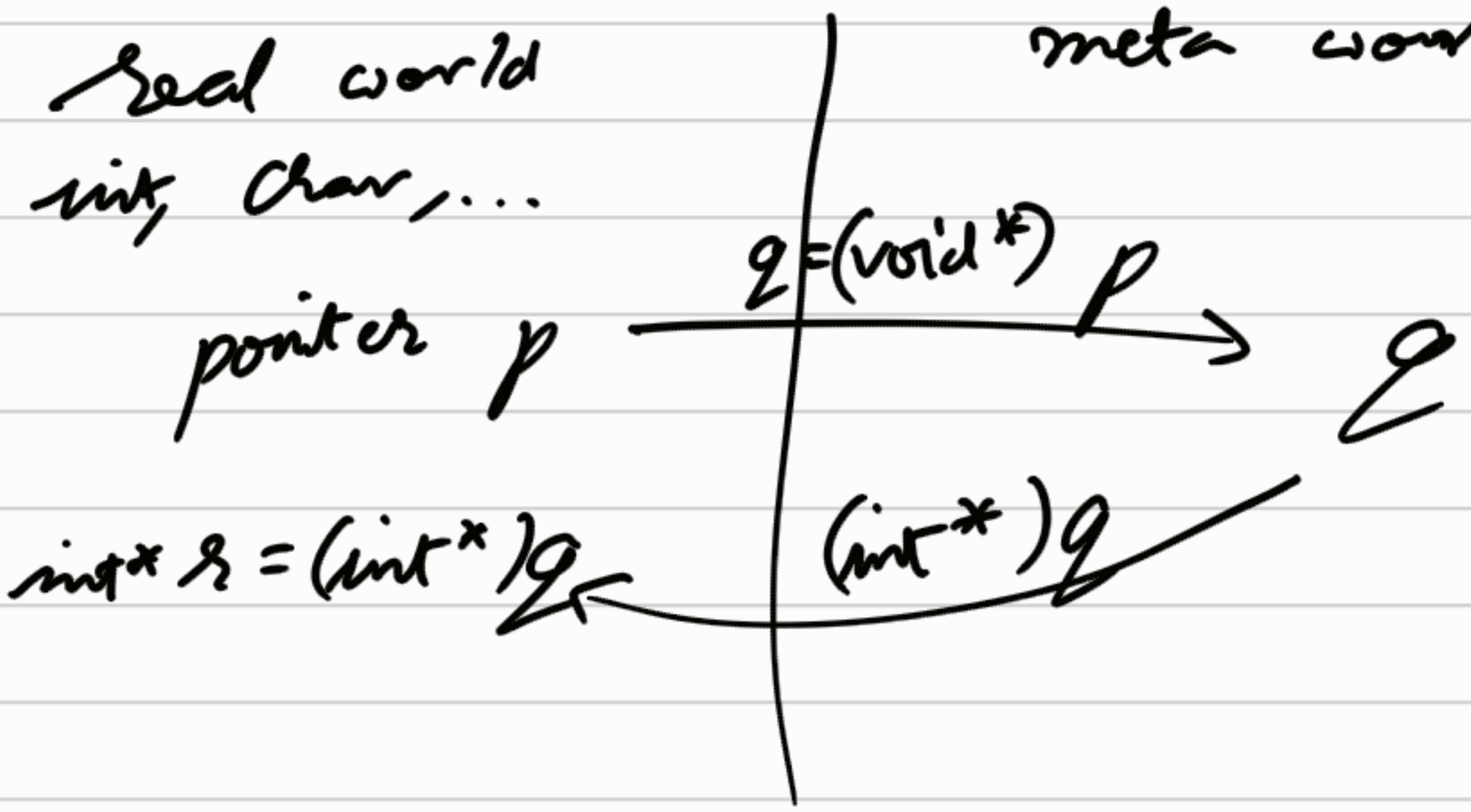
C1 :-
① generic pointers
② function pointers.

running C1 program :-
$ cc0 -d stack.c1 client/fib.c1
$ ./a.out

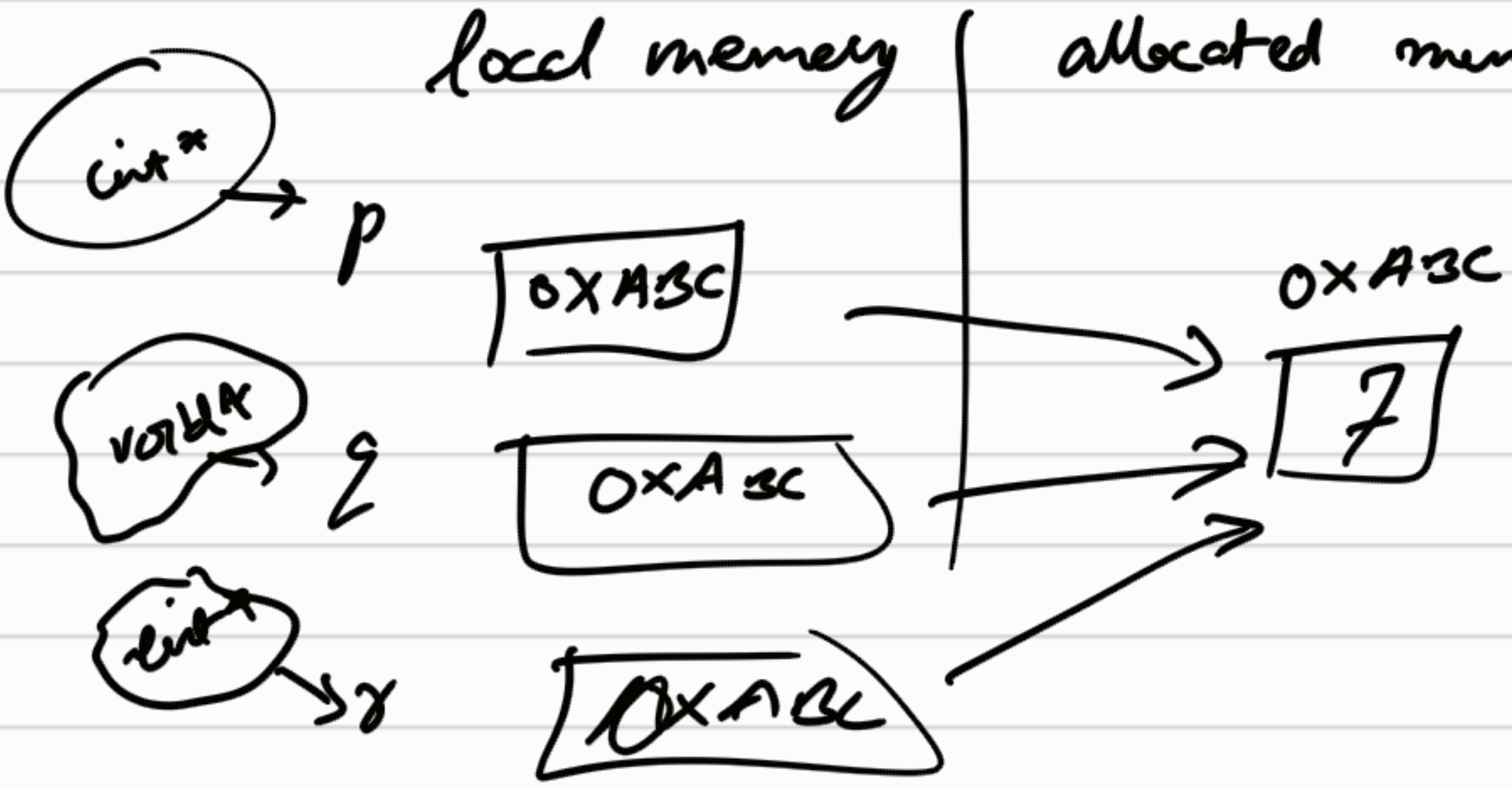
1. Defining :-
Void * q ;

2). any other pointer cast (or turned into) a Void *

int * p = alloc(int)
*p = 7 ;
void * q = (void*) p ;




```
int* p = alloc(int);
*p = 7
void* q = (void*) p;
int* r = (int*) q;
```



Operations on generic pointers.

Allowed ops

① cast to original type

② compare for equality

```
void* q1 = (void*) alloc(int);
if (q1 == q) ✓
```

③
```
void* q3 = q1;
int* p;
q = (void*) p
```

```
strij* s = (strij*) q   ✗✗ compilation error.
```

Error:

untagging pointer fail. ✓

Segmentation fault ✓

not allowed (illegal)

```
void x = *q;
```

Allocate:

```
void* a = alloc(void);
```

* cannot cast a generic pointer to some other type than the original.