

## Day 17 (Wed 3/2)

- day 17 recap questions
- exercise 17

## Announcements/reminders

- HW4 due Friday 3/4 at 11pm
  - written assignment, no late submissions
- Find a midterm project team and register ASAP!
  - See Piazza post 276
  - Contact me ASAP if you need help finding a team
- Midterm exam in class Friday 3/11
  - review materials are posted
  - exam format:
    - 60 minutes
    - computer based
    - access to editor/compiler/web resources

## Day 17 recap questions

1. Describe the linked list structure by a diagram.
2. Compare arrays and linked lists. Write down their pros and cons.
3. What is a linked list's head? How is it different from a node? Explain.
4. How do you calculate length of a linked list?
5. How do you implement `add_after` on a singly linked list?

1. Describe the linked list structure by a diagram.

```
struct LLNode {  
    char payload; // payload could be any type  
    struct LLNode *next;  
};
```

// code creating a linked list

```
Node *head = malloc(sizeof(struct Node));
```

```
head->payload = 'A';
```

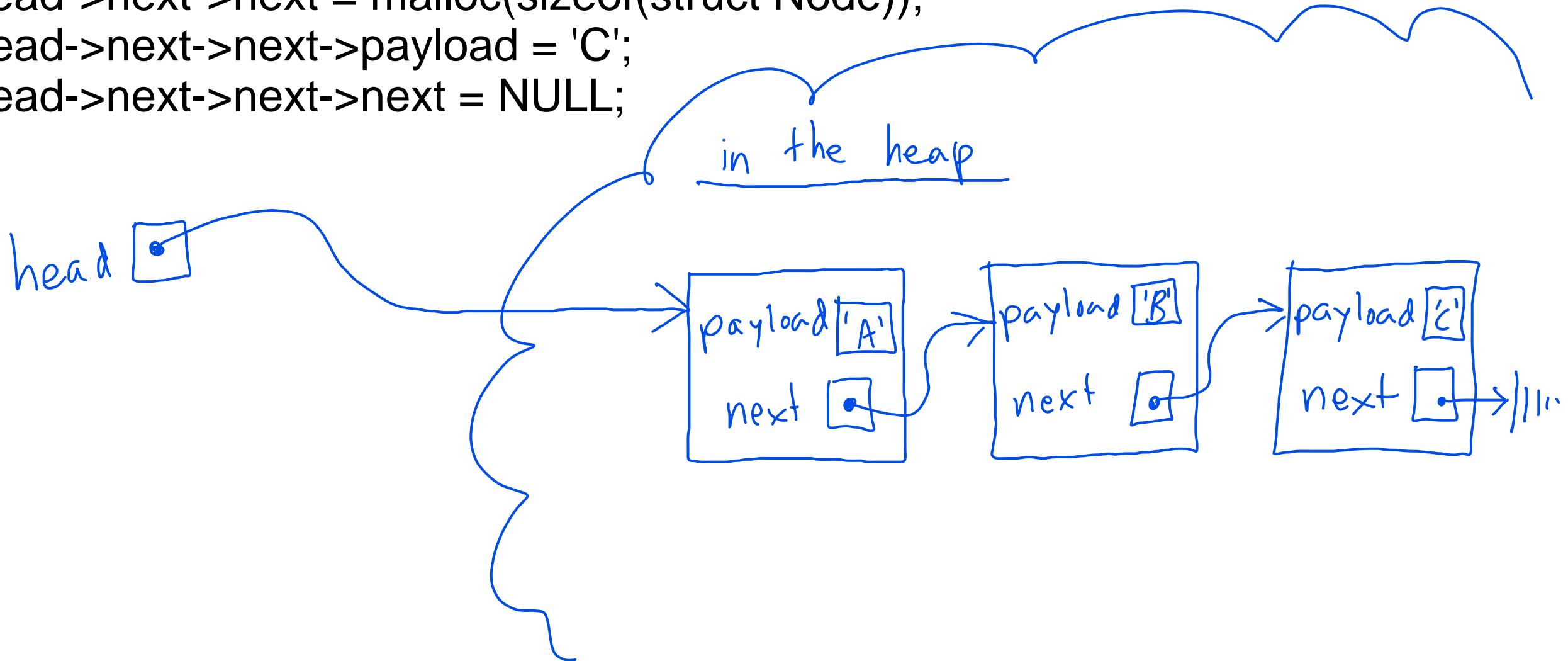
```
head->next = malloc(sizeof(struct Node));
```

```
head->next->payload = 'B';
```

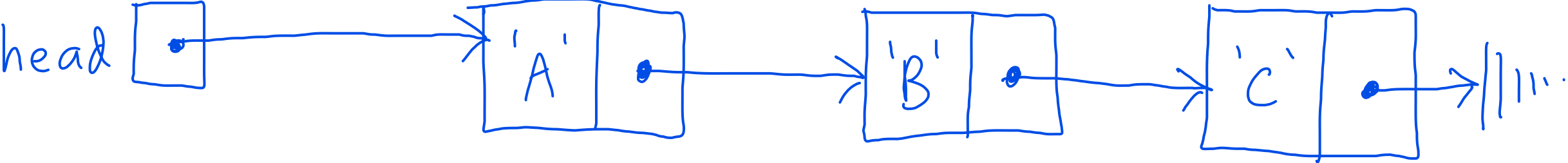
```
head->next->next = malloc(sizeof(struct Node));
```

```
head->next->next->payload = 'C';
```

```
head->next->next->next = NULL;
```



a more concise representation



2. Compare arrays and linked lists. Write down their pros and cons.

Arrays:

pro:  $O(1)$  access to arbitrary element

con:  $O(N)$  to insert or remove element at arbitrary position

pro: better locality (fewer cache misses when iterating)

pro: more compact

con: fixed size, to reallocate, must allocate new array and copy existing data

Linked lists:

con:  $O(N)$  access to arbitrary element

pro:  $O(1)$  to remove element at arbitrary position (given a pointer to predecessor)

con: worse locality (more cache misses when iterating)

con: less compact (next pointers require space)

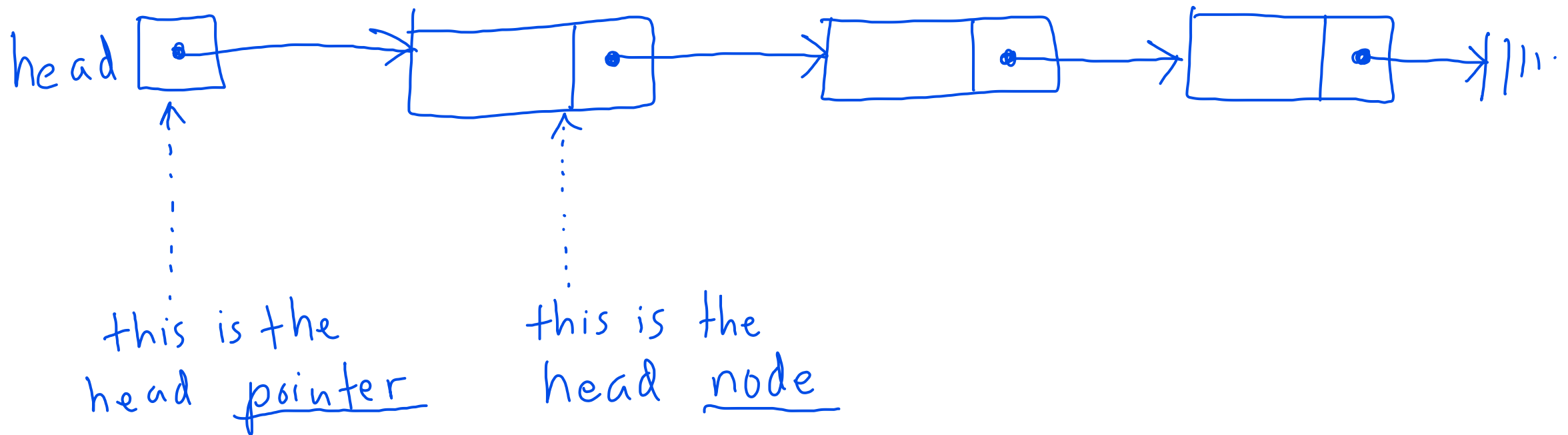
pro: can grow incrementally, nodes are allocated one at a time

3. What is a linked list's head? How is it different from a node? Explain.

Contrast: \*head pointer\* vs. \*head node\*.

The head pointer is a pointer to the first node in the list.

The head node \*is\* the first node in the list.



#### 4. How do you calculate length of a linked list?

A loop is required:

```
struct Node *head = /* points to first node in list */;  
int count = 0;
```

```
for (struct Node *cur = head; cur != NULL; cur = cur->next) {  
    count++;  
}
```

this is the way  
to advance to  
the next node  
in the list

loop ends  
when we reach  
the NULL pointer  
marking the end  
of the list



5. How do you implement add\_after on a singly linked list?

```
void add_after(struct Node *p, char value) {  
    struct Node *n = malloc(sizeof(struct Node));  
    n->payload = value;  
    n->next = p->next;  
    p->next = n;  
}
```

Trace:

