Week 8!

CLASS SEMESTER

Overview

Assignment 2

Malloc

Linked Lists

Struct Pointers

Assignment 2 Released

Assignment 2 is out

Any questions?

Assignment 2 livestream is the best place to go to get an easy overview on how to get started (Was live Tuesday in Matthews, recording up on the)

malloc

How much memory do we need for an int? How much for a double?

malloc

How much memory do we need for an int? How much for a double? (sizeof)

What about an array of 10 integers? (last week)

malloc

How much memory do we need for an int? How much for a double? (sizeof())

What about an array of 10 integers? (10 * sizeof(int))

What are the arguments and return of malloc (including return type)?

malloc

How much memory do we need for an int? How much for a double? (sizeof())

What about an array of 10 integers? (10 * sizeof(int))

What are the arguments and return of malloc? (void *malloc(size_t size))

Activity: malloc memory for all variables in the code on the following slide.
If you finish quickly, see if you can write code to change each of the values using only the pointer to the memory address the variable is stored at

```c
#include <stdio.h>

#define NUM_NUMS 100

int main(void) {
  int number;
  number = 3;
  double longer_number = 4 * number;
  char letter;
  int some_numbers[NUM_NUMS];

  for(int i = 0; i < NUM_NUMS; i++) {
    some_numbers[i] = i % 5 + i;
  }
  return 0;
}
```

When to choose malloc()

Using **malloc**:

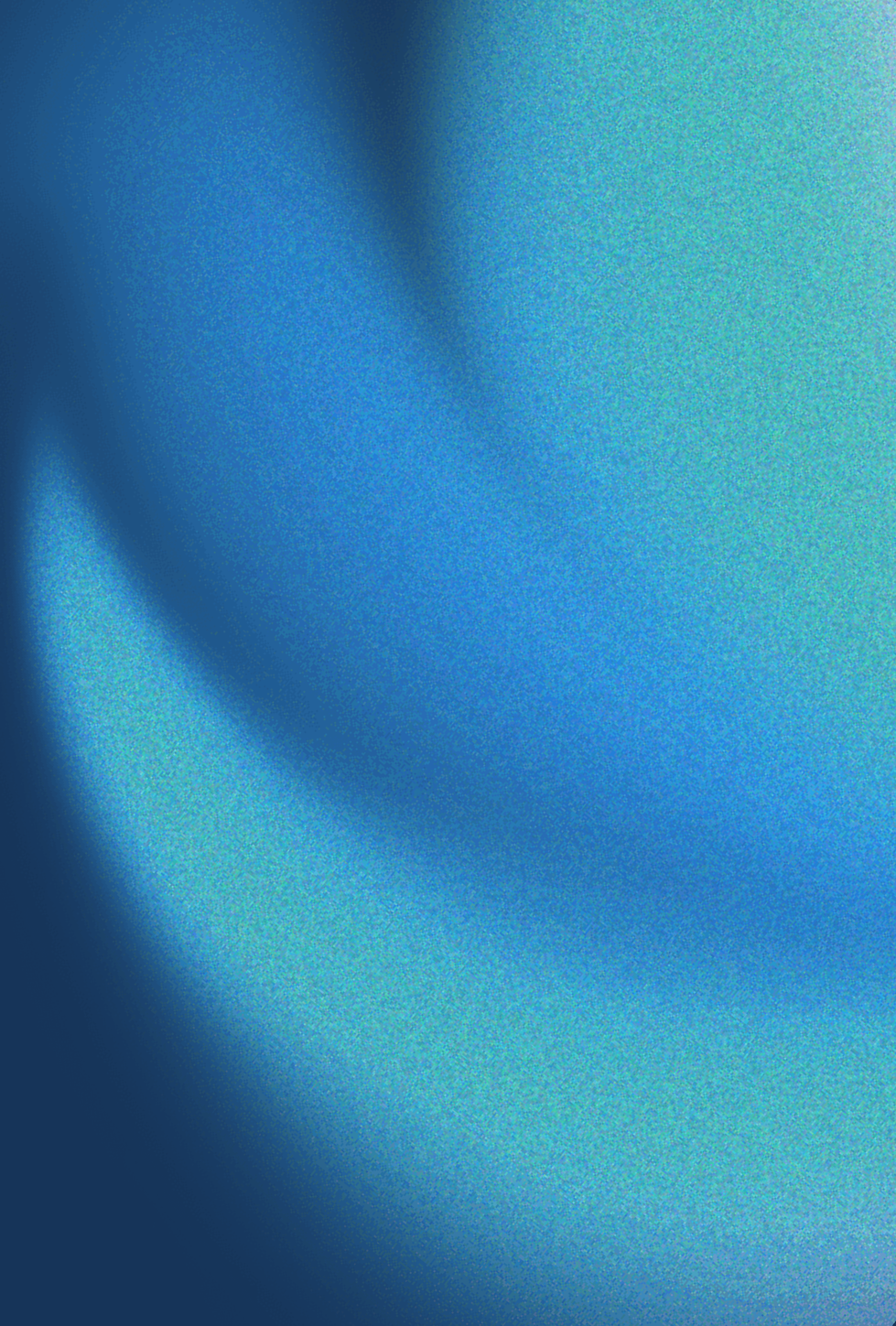
- dynamically choosing the size of our array (remember C doesn't like: int nums[size]; where size is scanned in at runtime)
- eventually we can decide to change the size of our array if that's required
- we can safely return a pointer towards the variable from a function

Not using **malloc**:

- simpler to use
- we don't need to worry about free()

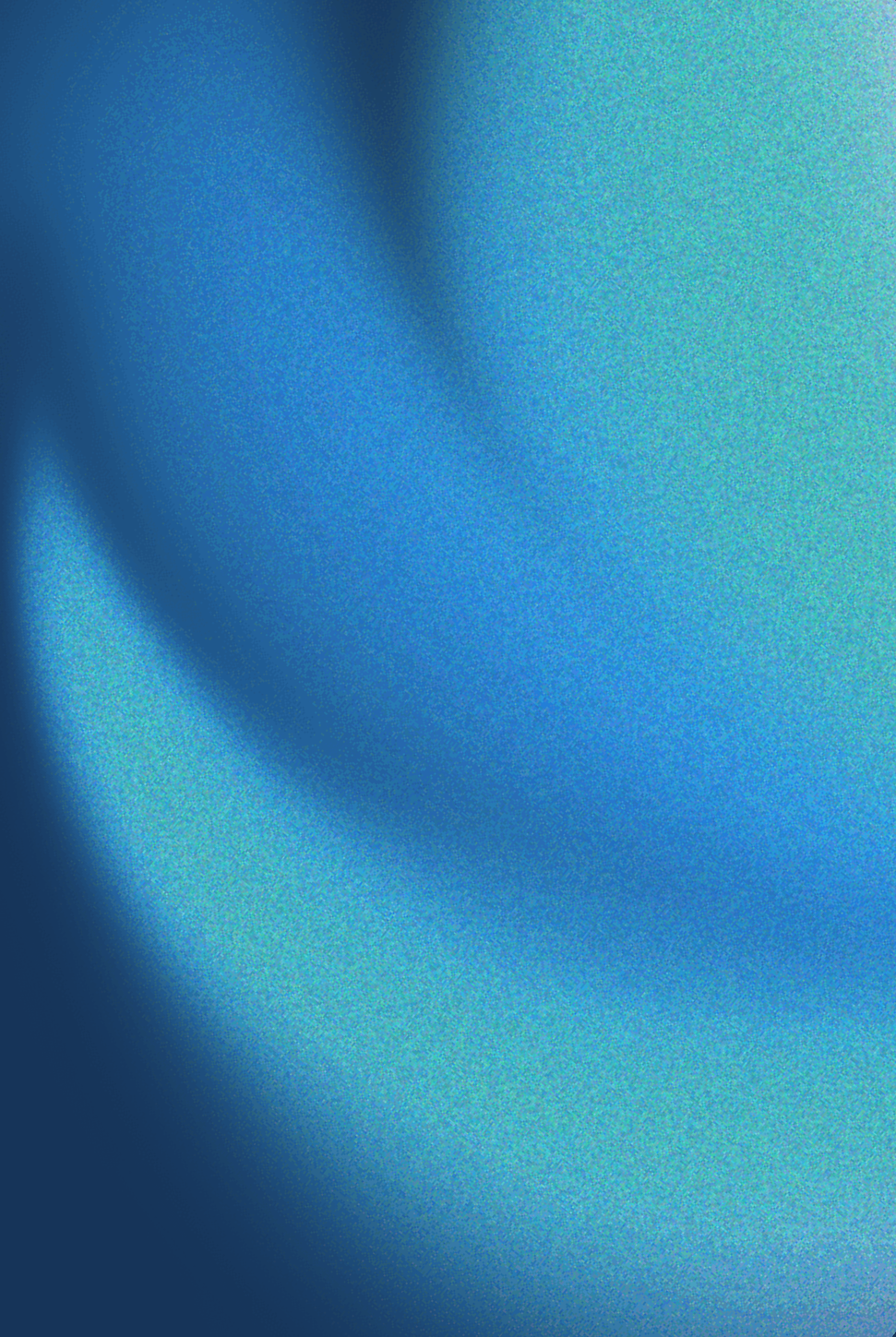
Let's malloc a struct

```
struct node {
    int data;
    struct node next;
}
```



Let's malloc a struct

```c
struct node {
    int data;
    struct node *next;
}
```



Tasks

1. Write up code to malloc a struct node
2. Write up code to initialise/populate the fields of a struct node
3. Move all of the above into a function so that it takes the values for it's fields (next should point to NULL) and returns a pointer to the new node

Diagramming Linked Lists

- Nodes linked together via next fields.
- Head pointer stores the address of the first node
- Need a current pointer too

Iterating through Linked List

Suggest some code

Iterating through Linked List

```
struct node *current = head;
while (current != NULL) {
    current = current->next;
}
```

Iterating through Linked List

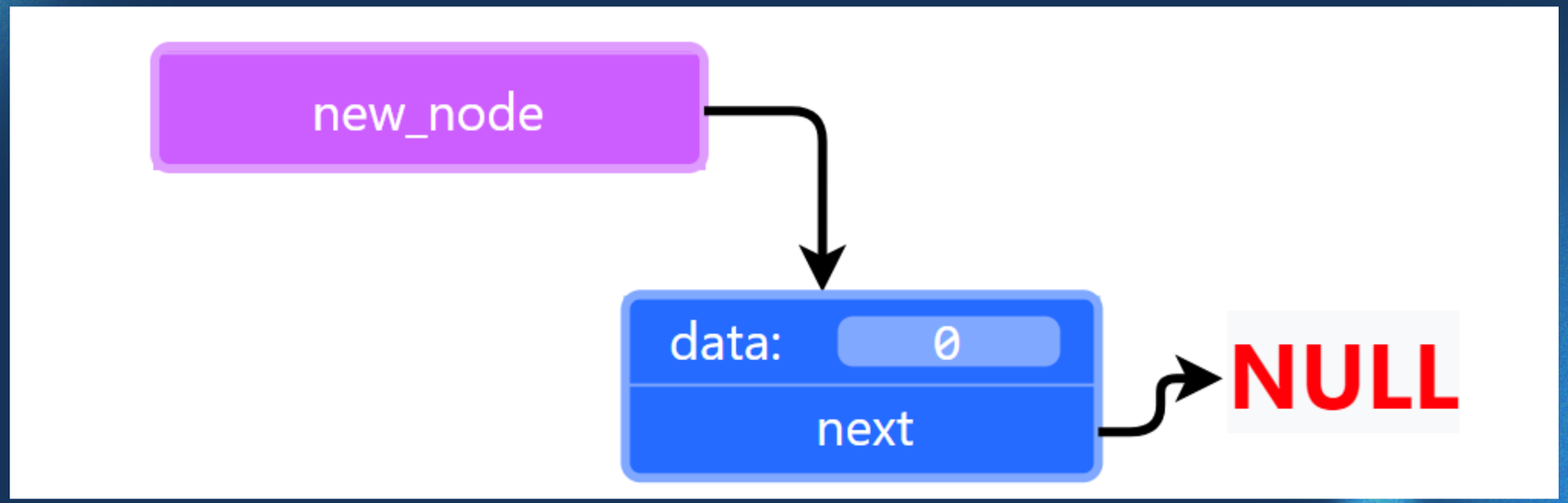
Suggest some code to deal with the case when the list is empty

```
struct node *current = head;
while (current != NULL) {
    current = current->next;
}
```

Iterating through Linked List

```
if (head == NULL) {
    return;
}
struct node *current = head;
while (current->next != NULL) {
    current = current->next;
}
```

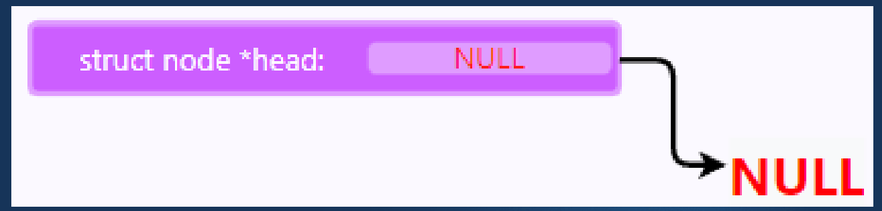
Struct Pointers

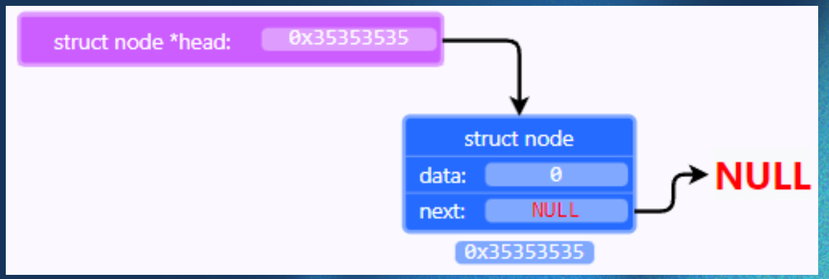


Task (Inserting into LL)

Every group gets a different edge case:

1. n = 0, and the list is any length (Diagram 1, 2 or 3).
2. n is greater than the length of the linked list.
3. n is any value, and the list is empty (Diagram 1).
4. n is less than the length of the list, and the list is not empty (Diagram 2, 3)





For your task write pseudocode for inserting before the nth node (handle your edge case and others if you have time)

Bonus: what order do we want to check our edge cases and why?