COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 11

Let’s go on a scavenger hunt (Linked Lists)

Multi-file projects

LAST TIME…

- Pointers
- A whole bunch of functions we can use with characters and strings: getchar(), putchar(), fgets(), etc.
- Strings
- Malloc'ing memory
- The one, the only, the truly magical, magnificent Linked Lists - a very basic think about it moment

TODAY...

- Let’s talk about the linked list
- Multi-file projects

WHERE IS THE CODE?

Live lecture code can be found here:

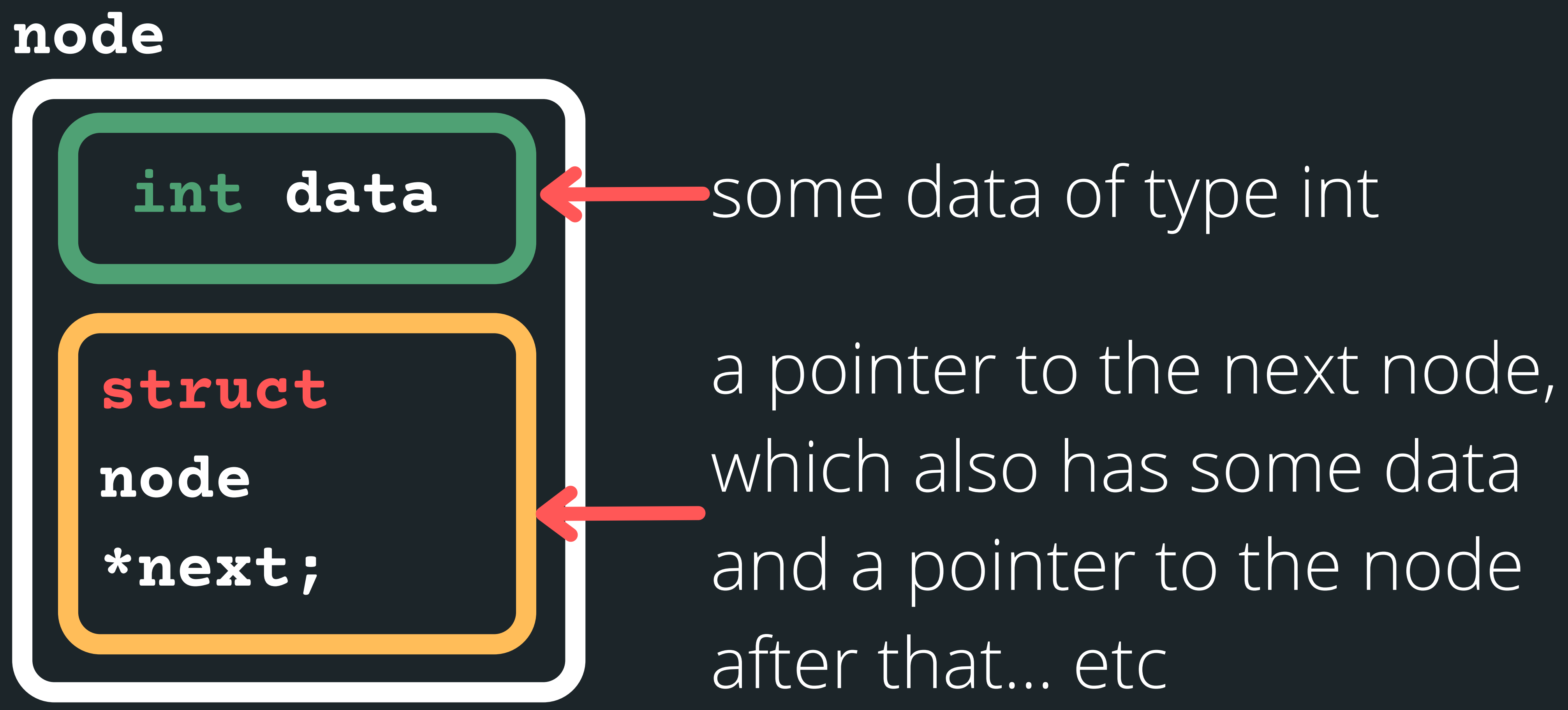
HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T1/LIVE/WEEK07/

A LINKED LIST IS MADE UP OF NODES

WHAT IS A NODE?

- Each node has some data and a pointer to the next node (of the same data type), creating a linked structure that forms the list
- Let me propose a node structure like this:

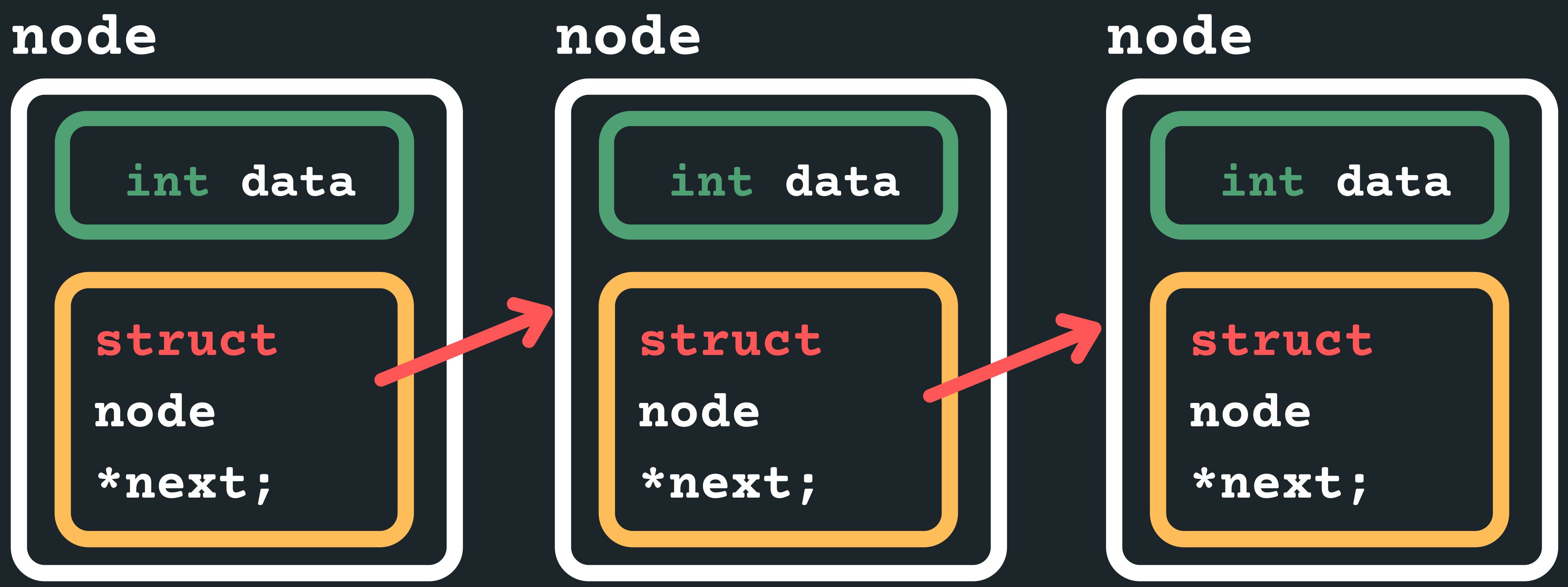
```c
struct node {
    int data;
    struct node *next;
};
```



A LINKED LIST IS MADE UP OF MANY NODES

THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- We can create a linked list, by having many nodes together, with each struct node next pointer giving us the address of the node that follows it

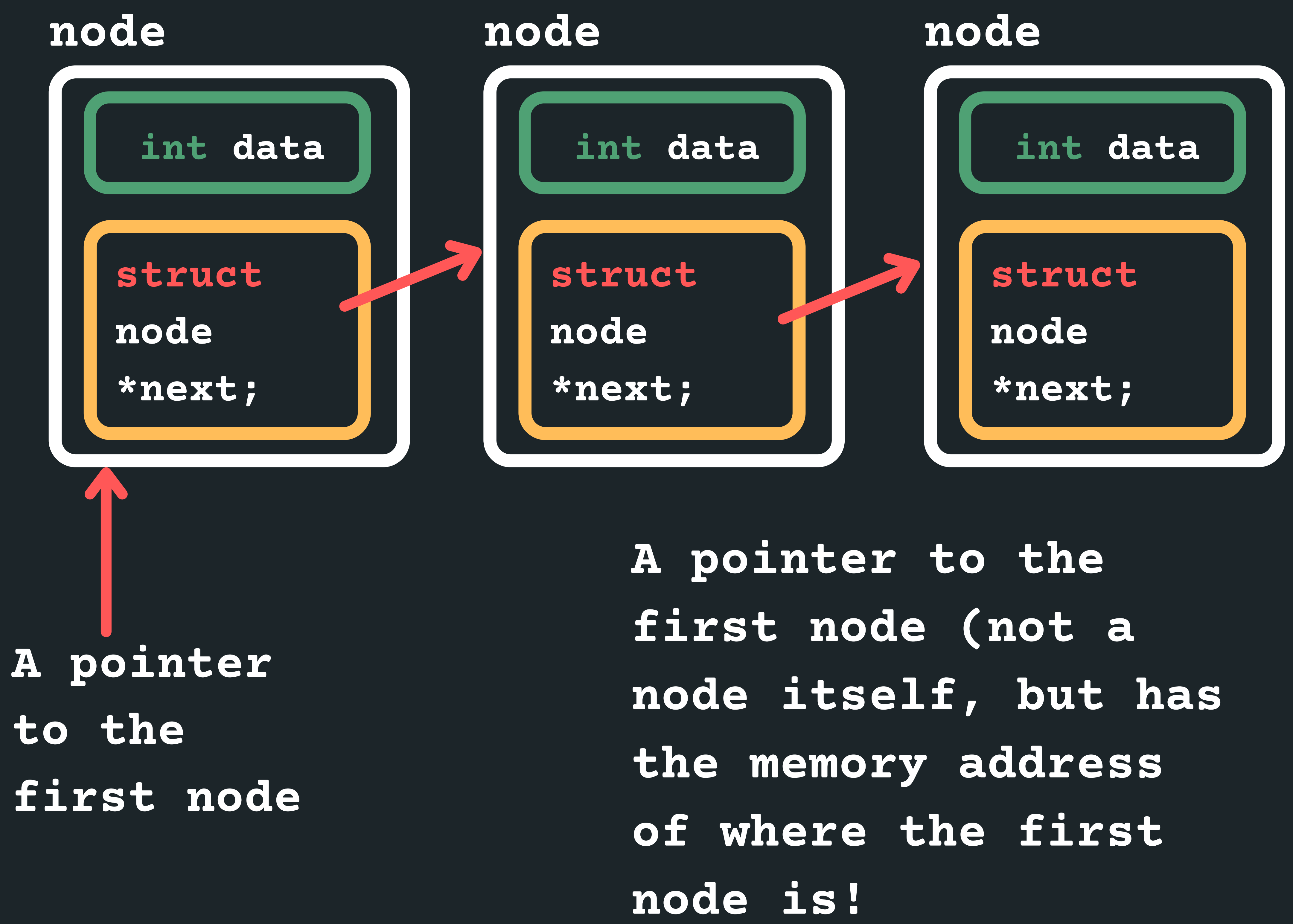


- But how do I know where the linked list starts?

A LINKED LIST IS MADE UP OF MANY NODES

THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- What about a pointer to the first node?

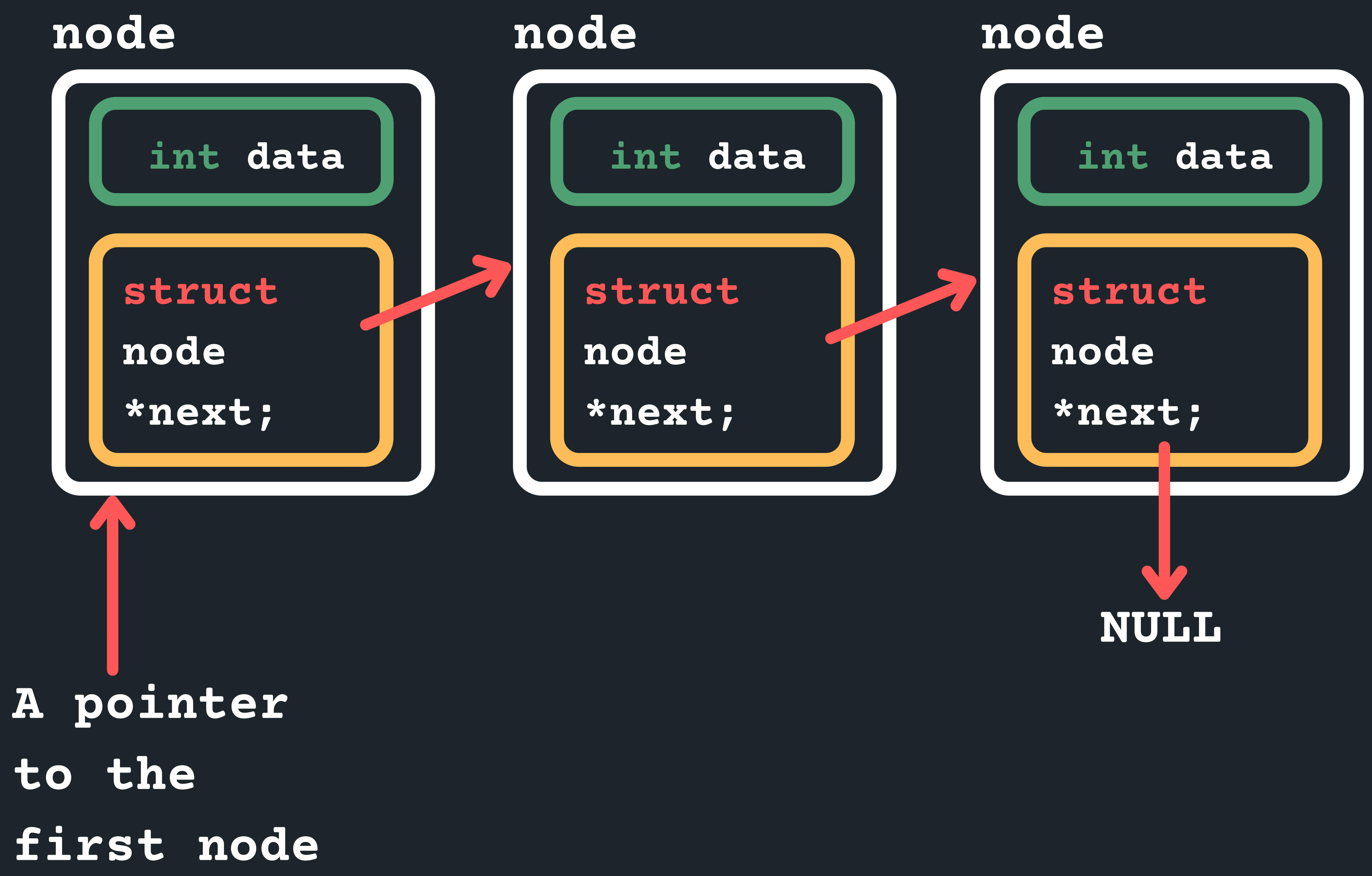


- How do I know when my list is finished?

A LINKED LIST IS MADE UP OF MANY NODES

THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

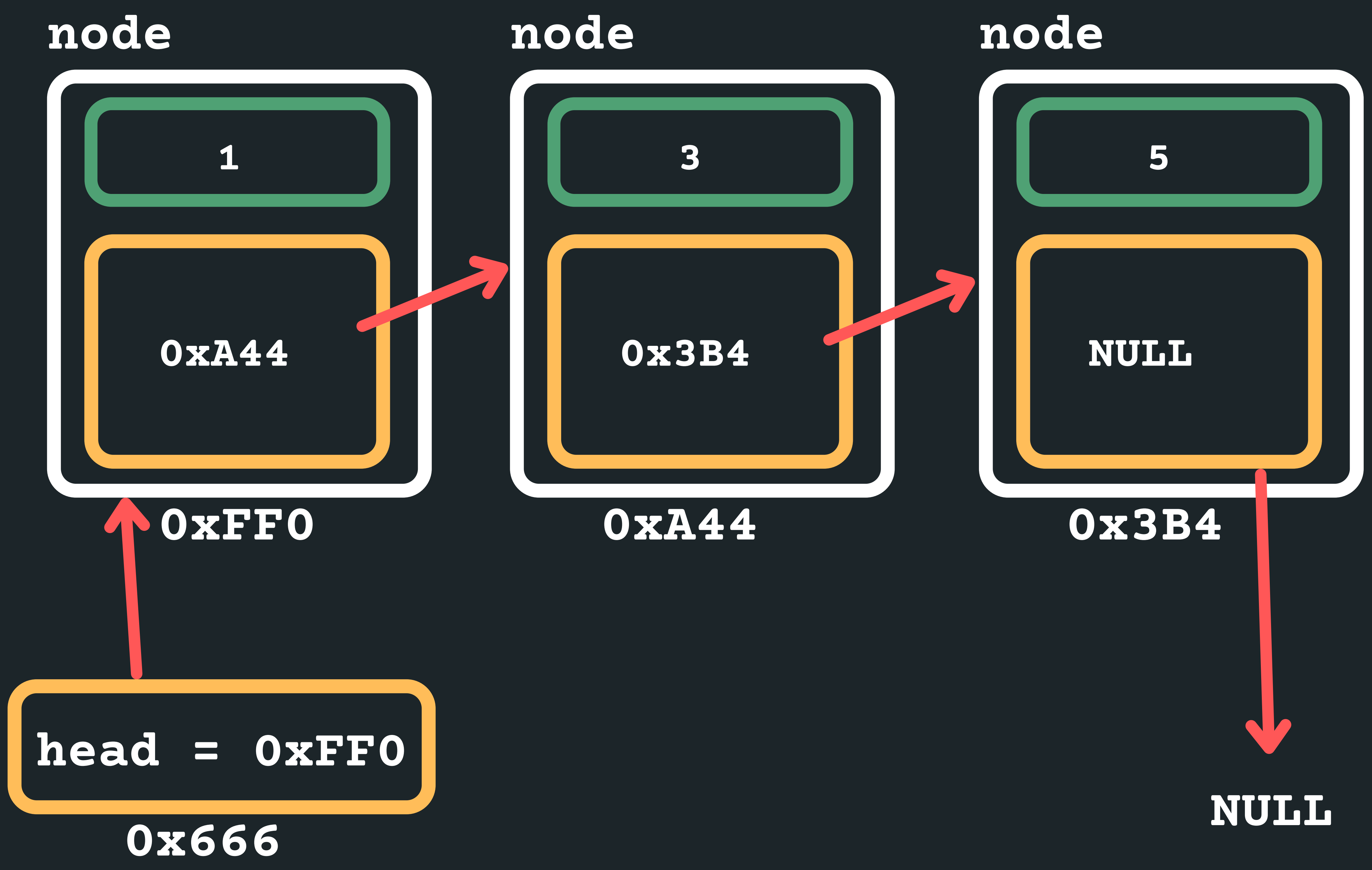
- Pointing to a NULL at the end!



A LINKED LIST IS MADE UP OF MANY NODES

THE NODES ARE LINKED TOGETHER (A SCAVENGER HUNT OF POINTERS)

- For example, a list with: 1, 3, 5



A LINKED LIST

WHY?

- Linked lists are dynamically sized, that means we can grow and shrink them as needed - efficient for memory!
- Elements of a linked list (called nodes) do NOT need to be stored contiguously in memory, like an array.
- We can add or remove nodes as needed anywhere in the list, without worrying about size (unless we run out of memory of course!)
- We can change the order in a linked list, by just changing where the next pointer is pointing to!
- Unlike arrays, linked lists are not random access data structures! You can only access items sequentially, starting from the beginning of the list.

A LINKED LIST

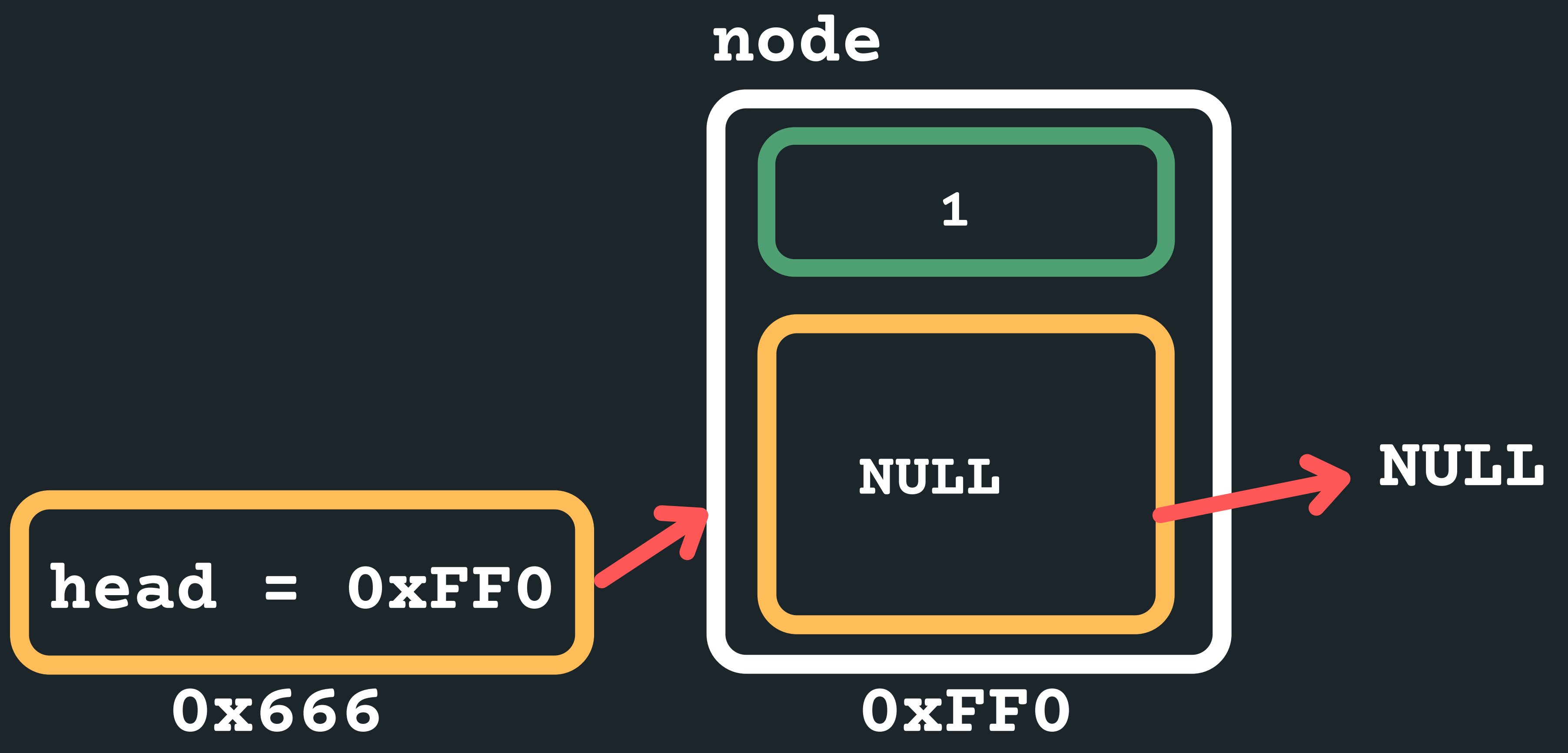
HOW DO WE CREATE ONE AND INSERT INTO IT?

- In order to create a linked list, we would need to
  - Define struct for a node,
  - A pointer to keep track of where the start of the list is and
  - A way to create a node and then connect it into our list…

A LINKED LIST

HOW DO WE CREATE ONE AND INSERT INTO IT?

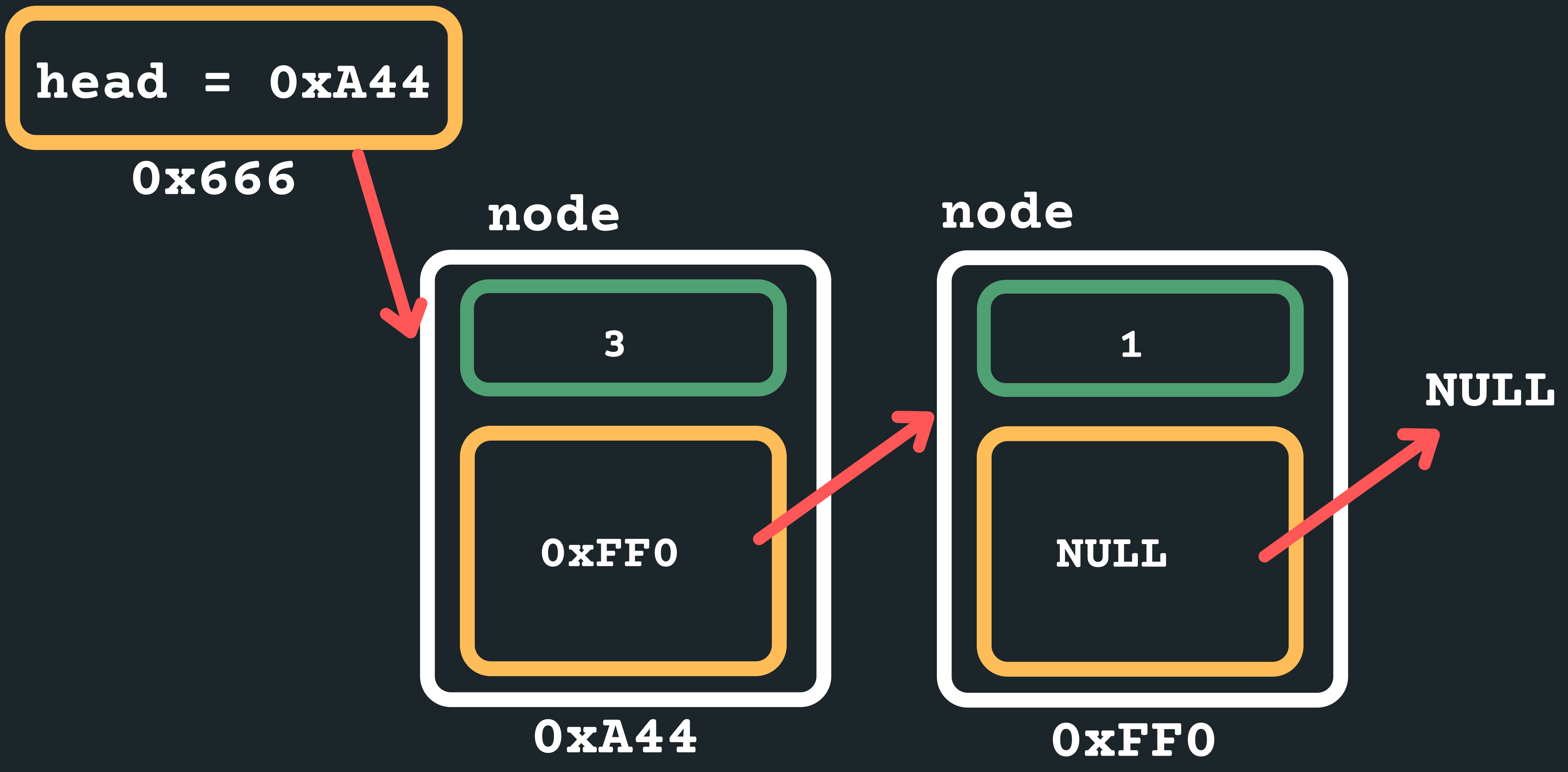
- Let’s say we wanted to create a linked list with 5, 3, 1
    - Let’s create the first node to start the list!
    - A pointer to keep track of where the start of the list is and by default the first node of the list
    - It will point to NULL as there are no other nodes in this list.



A LINKED LIST

HOW DO WE CREATE ONE AND INSERT INTO IT?

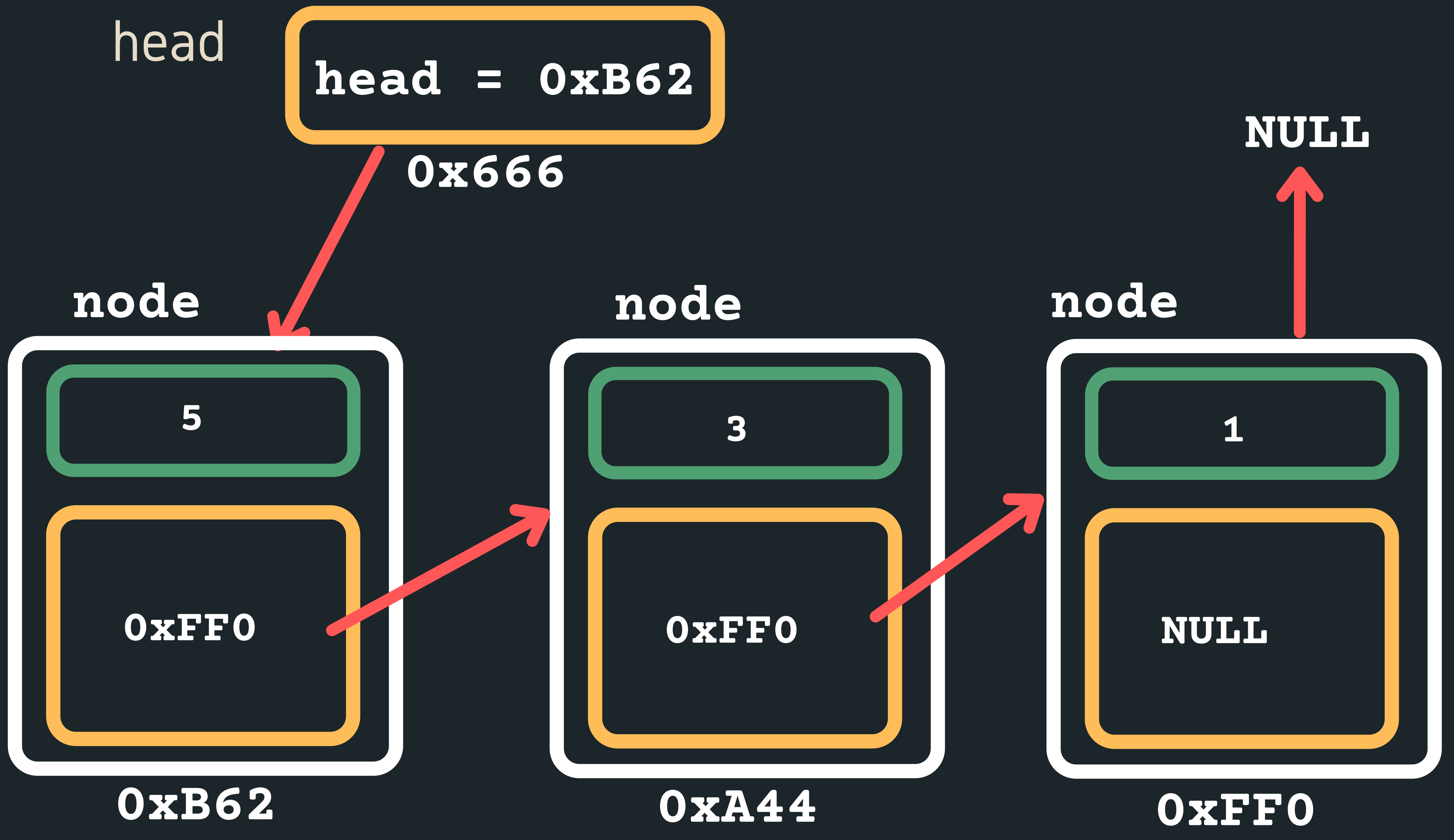
- Create the next node to store 3 into (you need memory)
- Assign 3 to data
- and insert it at the beginning so the head would now point to it and the new node would point to the old head



A LINKED LIST

HOW DO WE CREATE ONE AND INSERT INTO IT?

- Create the next node to store 5 into (you need memory)
- Assign 5 to data
- and insert it at the beginning so the head would now point to it and the new node would point to the old head



A LINKED LIST

PUTTING IT ALL TOGETHER IN CODE

1. Define our struct for a node
2. A pointer to keep track of where the start of the list is:
   - The pointer would be of type struct node, because it is pointing to the first node
   - The first node of the list is often called the 'head' of the list (last element is often called the 'tail')
3. A way to create a node and then connect it into our list...
   - Create a node by first creating some space for that node (malloc)
   - Initialise the data component on the node
   - Initialise where the node is pointing to
4. Make sure last node is pointing to NULL

BREAK TIME...

You have 8 sticks; 4 of them are of one length and the four remaining ones are a different length. Arrange the sticks so that they form 3 identical squares.

STRUCTS AND POINTERS

-> VERSUS .

- Remember that when we access members of a struct we use a .

```c
#include <stdio.h>
#include <string.h>

#define MAX 15

//1. Define struct
struct ice_cream {
    char name[MAX];
    double price;
};

int main (void) {
    //2. Declare struct
    struct ice_cream my_ice_cream;

    // 3. Initialise struct (access members with .)
    // Remember we can't just do my_ice_cream.name = "Dulce"
    // So we will use the function strcpy() in <string.h> to
    // copy the string over
    // strcpy(my_ice_cream.name, "Dulce");
    // my_ice_cream.price = 4.2;
    strcpy(my_ice_cream.name, "Dulce");
    my_ice_cream_ptr.price = 4.2;

    printf("%s is the best Messina flavour, and is $%.2lf\n", my_ice_cream.name,
                                                              my_ice_cream.price);

    return 0;
}
```

STRUCTS AND POINTERS

-> VERSUS .

- What happens if we make a pointer of type struct? How do we access it then?

```c
#include <stdio.h>
#include <string.h>

#define MAX 15

//1. Define struct
struct ice_cream {
    char name[MAX];
    double price;
};

int main (void) {
    //2. Declare struct
    struct ice_cream my_ice_cream;

    // Have a pointer that points to the variable my_ice_cream of type struct
    // ice_cream
    struct ice_cream *my_ice_cream_ptr = &my_ice_cream;

    // 3. Initialise struct (access members with .)
    // Remember we can't just do my_ice_cream.name = "Dulce"
    // So we will use the function strcpy() in <string.h> to copy the string over
    // strcpy(my_ice_cream.name, "Dulce");
    // my_ice_cream.price = 4.2;

    //How would we initialise it using the pointer?
    //Perhaps dereference the pointer and access the member?
    strcpy((*my_ice_cream_ptr).name, "Dulce");
    (*my_ice_cream_ptr).price = 4.2;

    printf("%s is the best Messina flavour, & is $%.2lf\n",(*my_ice_cream_ptr).name,
                                                            (*my_ice_cream_ptr).price);
    return 0;
}
```

STRUCTS AND POINTERS

-> VERSUS .

- Those brackets can get quite confusing, so there is a shorthand way to do this with an ->
- There is no need to use (*my_ice_cream_ptr) and instead can just straight my_ice_cream_ptr ->

```
    printf("%s is the best Messina flavour, & is $%.2lf\n",
                                                            (*my_ice_cream_ptr).name,
                                                            (*my_ice_cream_ptr).price);
```

shorthand way - much simpler to use
and harder to make mistakes
when you use ->

```
    printf("%s is the best Messina flavour, & is $%.2lf\n",
                                                            my_ice_cream_ptr->name,
                                                            my_ice_cream_ptr->price);
```

MULTI FILE PROJECT

WHAT ARE THEY?

- Big programs are often spread out over multiple files. There are a number of benefits to this:
  - Improves readability (reduces length of program)
  - You can separate code by subject (modularity)
  - Modules can be written and tested separately
- So far we have already been using the multi-file capability. Every time we #include, we are actually borrowing code from other files
- We have been only including C standard libraries

MULTI FILE PROJECT

WHAT ARE THEY?

- You can also #include your own! (FUN!)
- This allows us to join projects together
- It also allows multiple people to work together on projects out in the real world
- We will also often produce code that we can then use again in other projects (that is all that the C standard libraries are - functions that are useful in multiple instances)

MULTI FILE PROJECT INCLUDES

.H FILE
.C FILE (MAYBE MULTIPLES)

- In a multi file project we might have:
- (multiple) header file - this is the .h file that you have been using from standard libraries already
- (multiple) implementation file - this is a .c file, it implements what is in the header file.
- Each header file that you write, will have its own implementation file
- a main.c file - this is the entry to our program, we try and have as little code here as possible

header file

#include "   .h"

implementation file

.c

HEADER FILE

#INCLUDE "SOMETHING.H"

Typically contains:

- function prototypes for the functions that will be implemented in the implementation file
- comments that describe how the functions will be used
- #defines
- the file basically SHOWS the programmer all they need to know to use the code
- NO RUNNING CODE
- This is like a definition file

IMPLEMENTATION FILE

SOMETHING.C

This is where you implement the functions that you have defined in your header file

IMPLEMENTATION FILE

MAIN.C

This is where you call functions from that may exist in other modules.

AN EXAMPLE

A MATHS

- We will have three files:
  - header file - maths.h
  - implementation file - maths.c
    - #include ”maths.h”
  - main file - main.c
    - #include ”maths.h”

*maths.h

```c
// This is the header file for the maths module example
// The header file will contain:
//      - any #define
//      - function prototypes and any comments

#define PI 3.14

//Function prototype for a function that calculates
//square of a number
int square(int number);

//Function prototype for a function that calculates
//sum of two numbers
int sum(int number1, int number2);
```

main.c

```c
//This is the main file in our program
//This is where we drive the program from and where we
//make calls to our modules. We need to include the
//header file for each module that we want to use functions
//from

#include <stdio.h>
//Include the header file:
#include "maths.h"

int main (void) {
    int number = 13;
    int number2 = 10;

    printf("The square of the number %d is %d\n", number, square
(number));
    printf("The sum of %d and %d is %d\n", number, number2, sum
(number, number2));
    return 0;
}
```

maths.c

```c
//This is the implementation file of maths.h
//We defined two functions in the header file,
//and this is where we will implement these two
//functions

//Include your header file in the implementation file
//by using the below syntax

#include "maths.h"

int square(int number) {
    return number * number;
}

int sum(int number1, int number2) {
    return number1 + number2;
}
```

COMPILING A MULTI FILE

COMPILE ALL C FILES IN THE PROJECT

- To compile a multi file, you basically list any .c files you have in your project
  - In the case of our example, we have a maths.c and a main.c file):

```
Terminal
File Edit View Terminal Tabs Help
avas605@vx3:~/maths_module$ dcc maths.c main.c -o maths
avas605@vx3:~/maths_module$ ./maths
The square of the number 13 is 169
The sum of 13 and 10 is 23
avas605@vx3:~/maths_module$
```

- The program will always enter in main.c, so there should only be one main.c when compiling

Feedback please!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://www.menti.com/7ddzvez5py

WHAT DID WE LEARN TODAY?

LINKED LISTS

linked_list.c

main.c

linked_list.h

MULTI-FILE PROJECTS

maths.c

main.c

maths.h

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

@

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au