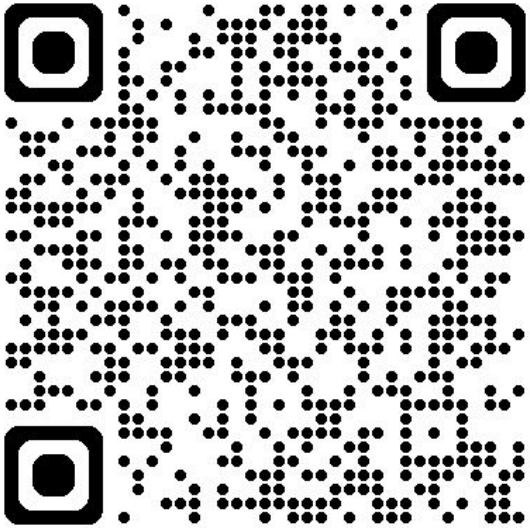
COMP1511/1911 Programming Fundamentals

Week 7 Lecture 2

Linked Lists

Link to Week 7 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T3/code/week_7/



Next Week: Week 8 Lab Exam

- In your lab time
- To prepare you for the format of the final invigilated exam
- Please attend your week 8 lab as scheduled
  - Online classes will get more information ASAP
- Worth 1 lab mark
- Email course account if for some reason you are sick or can’t attend on the day

Week 8 Lab Marks

- 0.5 1 dot
- 0.5 2 dot
- 0.5 3 dot
- 1 mark lab exam - attempt and submit a solution for at least 1 question
- Number of questions
  - 4
  - 2 array hurdles
  - 2 debugging

Next Week: Week 8 Revision Sessions

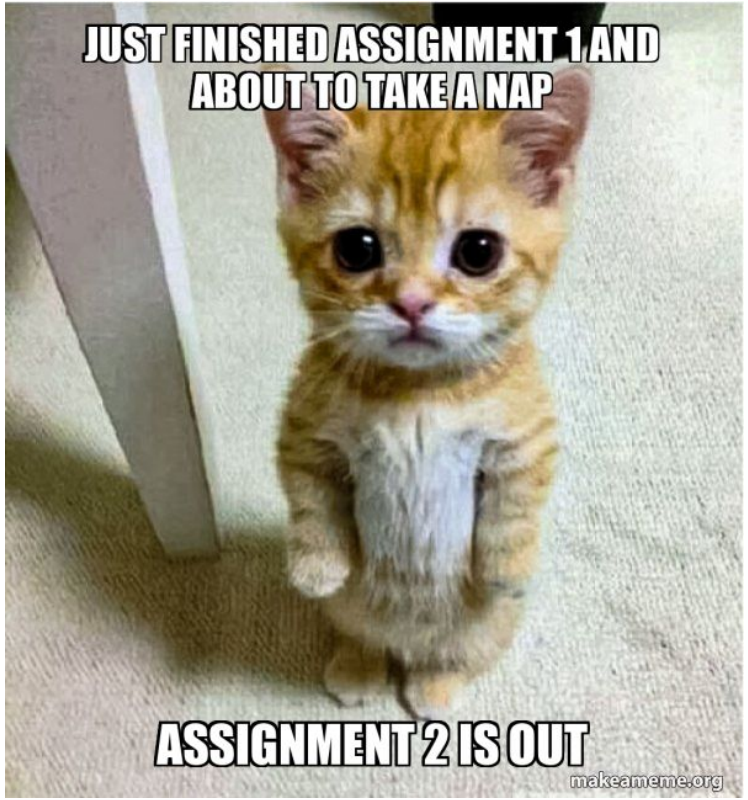
- Revision sessions in week 8
- Keep eye on forum for time, location, bookings and voting

Assignment 2

Time: Thursday 9am

Assignment Due Date:
Friday Week 10 5pm

Don’t leave it until the last minute!
Help sessions will be very busy the week before the deadline!!!!!!!!



Assignment 2

- It is an individual assignment
- Aims of the assignment
  - Work with a larger problem and codebase
  - Work with multiple C files
  - Problem solve with linked lists
  - Practice using strings
  - Being a responsible heap user (free your malloced memory)
- You will be assessed on style! 20% of your mark
- COMP1911 only need to complete up to and including stage 3.3

Assignment 2 Linked List Warning

You **MUST** use linked linked lists.

- You can’t change the linked lists into arrays and just do it with arrays!!!!!!!!
- You will get 0 performance in the assignment if you do

Last Lecture

- Pointers basics recap
- Pointers and arrays
- Memory and the stack
- Dynamic Memory, malloc and the heap
- Multi-file projects

Today's Lecture: Linked Lists

- Why are we learning linked lists?
- What is a linked list?
- Inserting at the head
- Traversing a linked list
- Inserting at the tail

But first a quick recap of malloc and a quick look at realloc

Using malloc

- multiply the number of elements you need by the sizeof the type of the element to work out how many bytes you want malloc to give you
- malloc will return a pointer to the starting address of the chunk of memory it allocated

```
int *numbers = malloc(10 * sizeof(int));
```

Putting it all together

```c
// create array
int *data = malloc(num_elements *sizeof(int));
// check malloc was successful

// Use the array somehow
// etc etc

// Free array when finished with array
free(data);
```

Note: You can check for memory leaks using dcc with the flag

`dcc --leak-check`

The realloc function

- What happens if this array wants to actually grow after you have filled it up?
- We can use realloc on a dynamically created array

```
// Ask malloc for enough memory for an array of 10 ints
int *numbers = malloc(10 * sizeof(int));

// Decide later we need enough for 20 ints
// It will make the array bigger, without destroying the
// contents
numbers = realloc(numbers, 20 * sizeof(int));
```

Realloc coding example

realloc_array.c

Exercise: return pointer to struct

```
struct person {
    char name[MAX_LEN];
    int age;
};
// return a pointer to a person struct with name and age
struct person *create_person(char *name, int age);
void print_person(struct person the_person);
```

Write the functions and write a main function to

1. Call the first function with “Tina Arena” and age 58
2. Call the function to print the person’s details

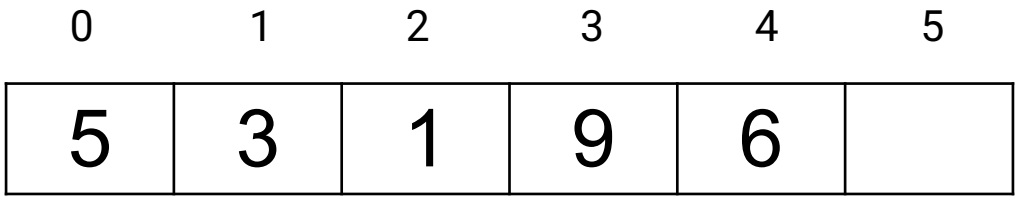
Linked Lists

Linked Lists

- An alternative to using an array to store collections of data
  - Arrays are amazing and we won't be forgetting about them
  - This is just another option!
- Linked Lists are suitable for sequential data:
  - playlists of songs
  - image galleries
  - web browser history
- Why would we want to use a linked list instead of an array?

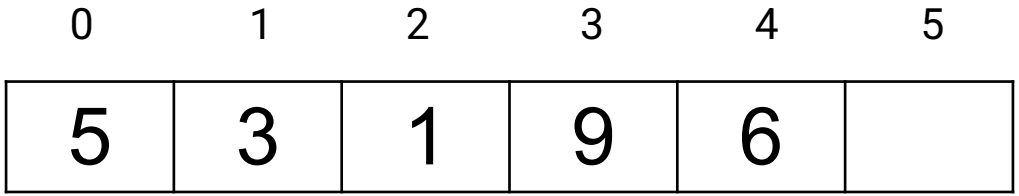
Array Advantages

- Store collections of data in contiguous blocks of memory
- Great for sequential access or random access
- It is easy to insert or delete items at the end



Array Disadvantages

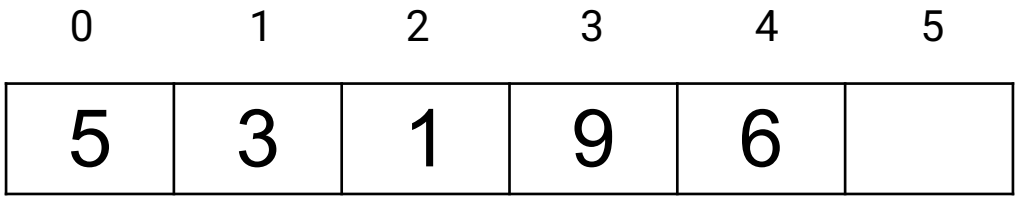
- Messy and inefficient for inserting or deleting in the middle
- E.g. How can we insert an item at or delete from index 1 in the array below?



Array Disadvantages

We would need to move all the subsequent data along to

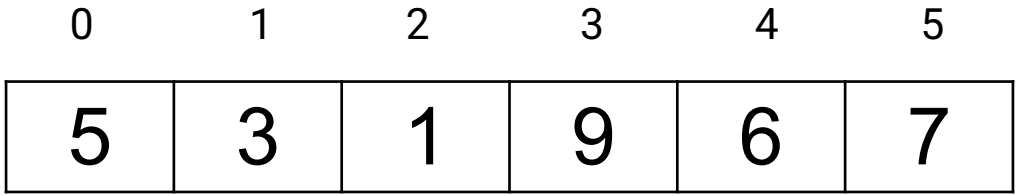
- make room to insert an item at index 1
- remove the gap if we deleted an item at index 1



Array Disadvantages

How can we insert an item into the array below?

- With a static array we can’t!
- With a dynamic array we can use realloc
  - How much bigger do we make it? Just 1 bigger? double the size?



Linked List Advantages

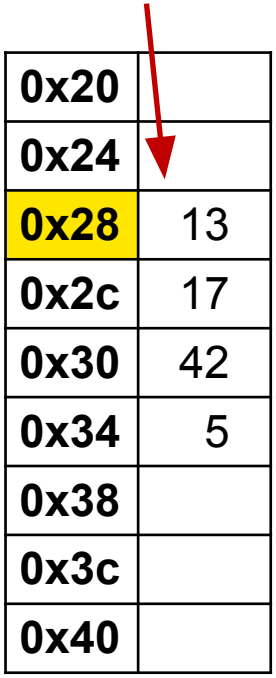
- They are dynamic structures
  - They grow and shrink as needed
- They don’t need contiguous memory like an array
- Insert or delete items anywhere in the list
  - by modifying one or two pointers
  - without moving existing data

Linked List Disadvantages

- Not good for random access 😕
  - You have to traverse from the beginning of the list
- Extra overhead of storing a pointer for each data item

Arrays in Memory

```
int array[] = {13, 17, 42, 5};
```



0x20	
0x24	
0x28	13
0x2c	17
0x30	42
0x34	5
0x38	
0x3c	
0x40	

- The array name gives us the address of the beginning of the chunk of memory
- Arrays are stored contiguously which allows us to use indexes and make random access quick and easy

Arrays vs Linked Lists in Memory

```
int array[] = {13, 17, 42, 5};
```

0x20	
0x24	
0x28	13
0x2c	17
0x30	42
0x34	5
0x38	
0x3c	
0x40	

- Linked list data is not contiguous
- It is scattered throughout memory.

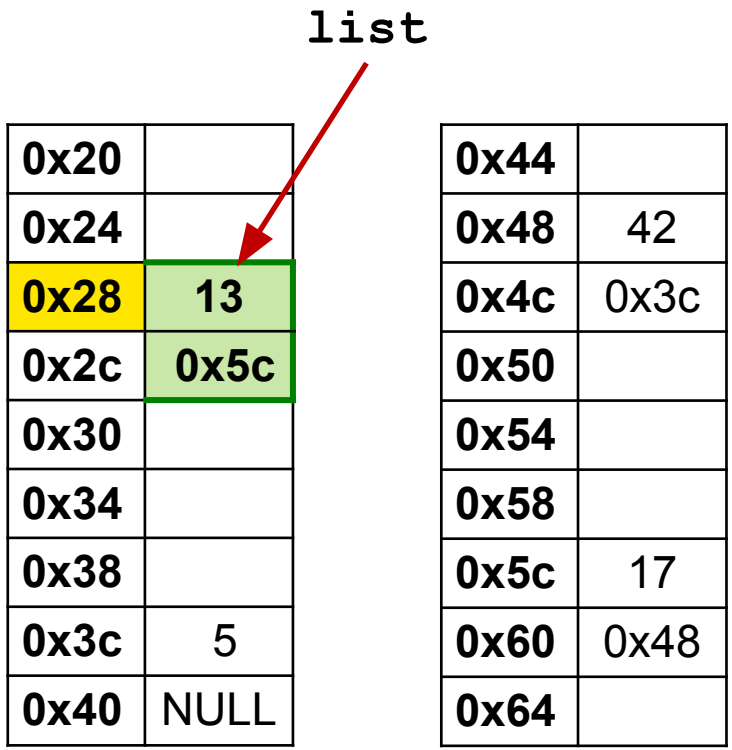
`list`

0x20	
0x24	
0x28	13
0x2c	0x5c
0x30	
0x34	
0x38	
0x3c	5
0x40	NULL

0x44	
0x48	42
0x4c	0x3c
0x50	
0x54	
0x58	
0x5c	17
0x60	0x48
0x64	

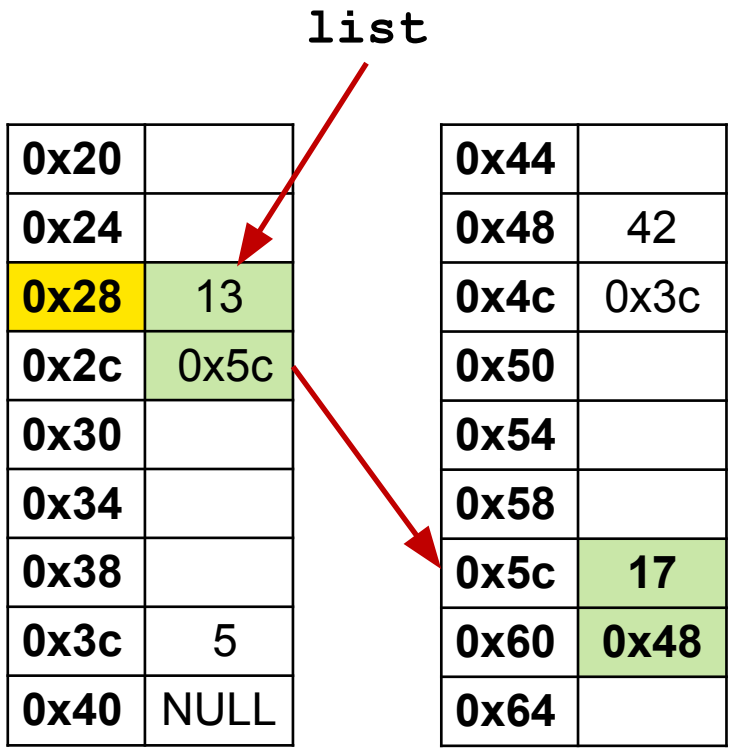
Linked Lists in Memory

- You need a pointer to the first piece of data in the list
- And for every piece of data you store in the list you need to store a link (pointer containing the address) to the next item in the list.
- Like a scavenger hunt.



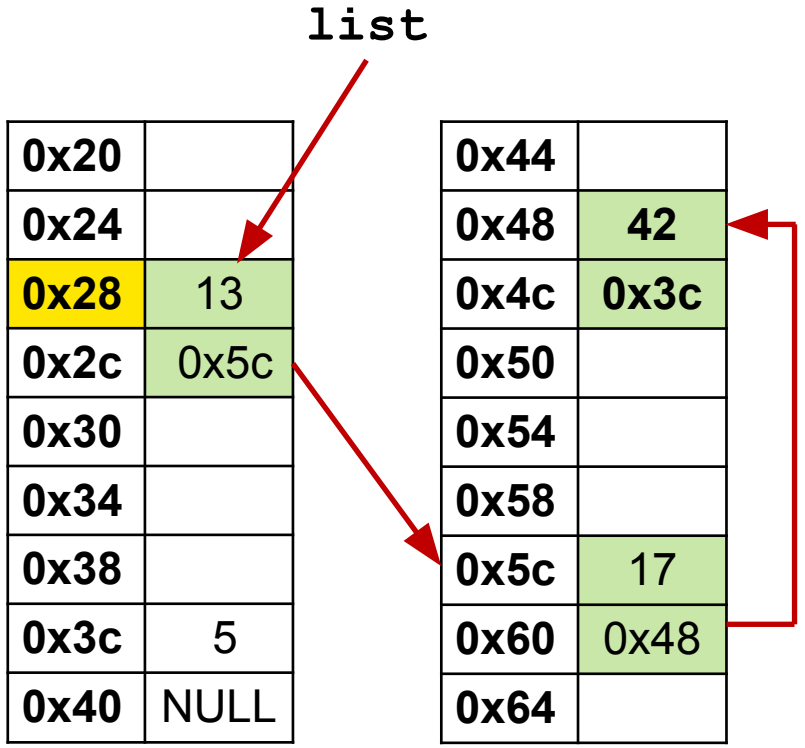
Linked Lists in Memory

- You need a pointer to the first piece of data in the list
- And for every piece of data you store in the list you need to store a link (pointer containing the address) to the next item in the list.
- Like a scavenger hunt.



Linked Lists in Memory

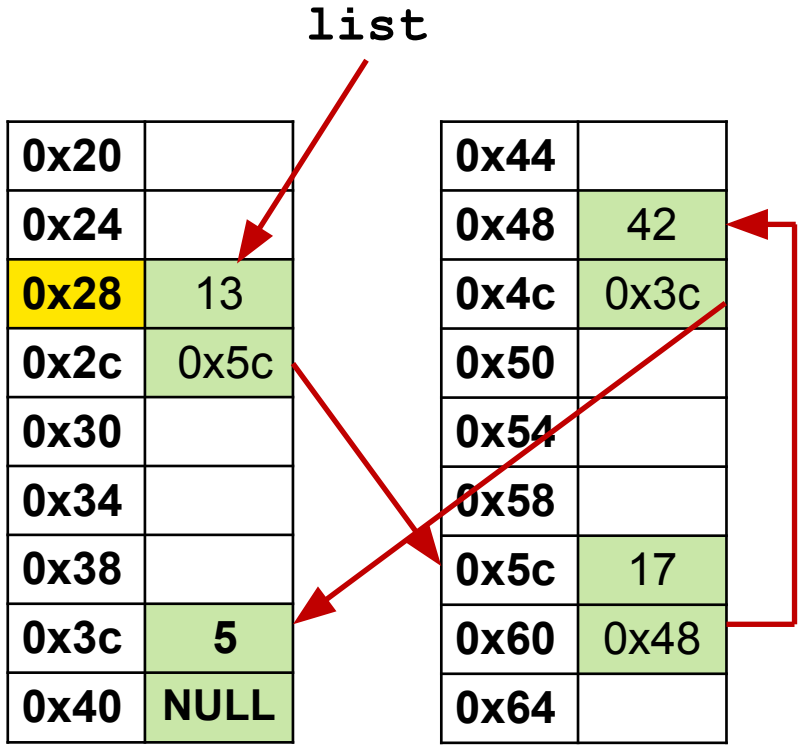
- You need a pointer to the first piece of data in the list
- And for every piece of data you store in the list you need to store a link (pointer containing the address) to the next item in the list.
- Like a scavenger hunt.



Linked Lists in Memory

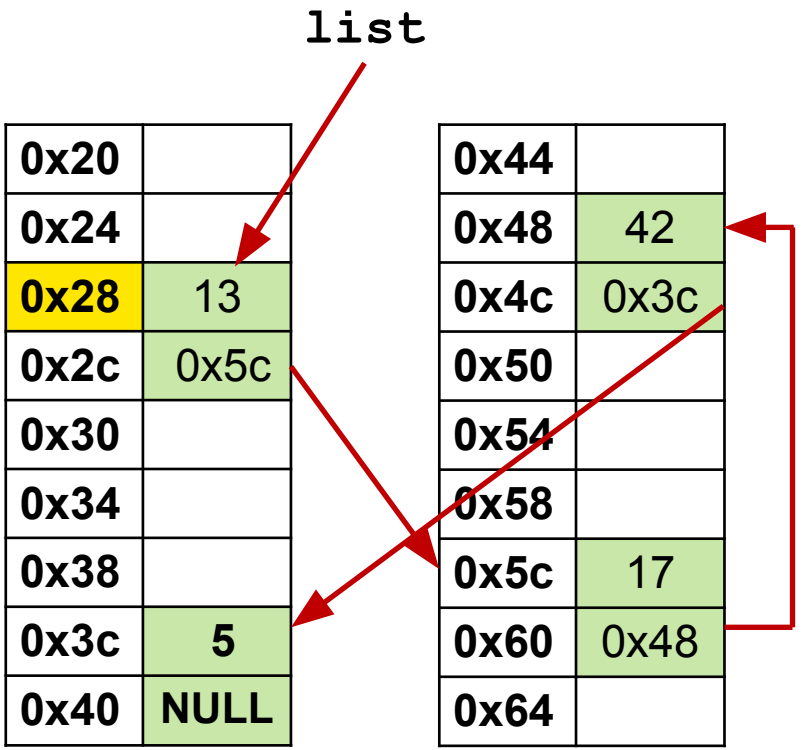
- When the value of the pointer to the next piece of data is NULL you have reached the end of the list.

- **Congratulations!**
  You have just traversed your first linked list.



Linked Lists in Memory

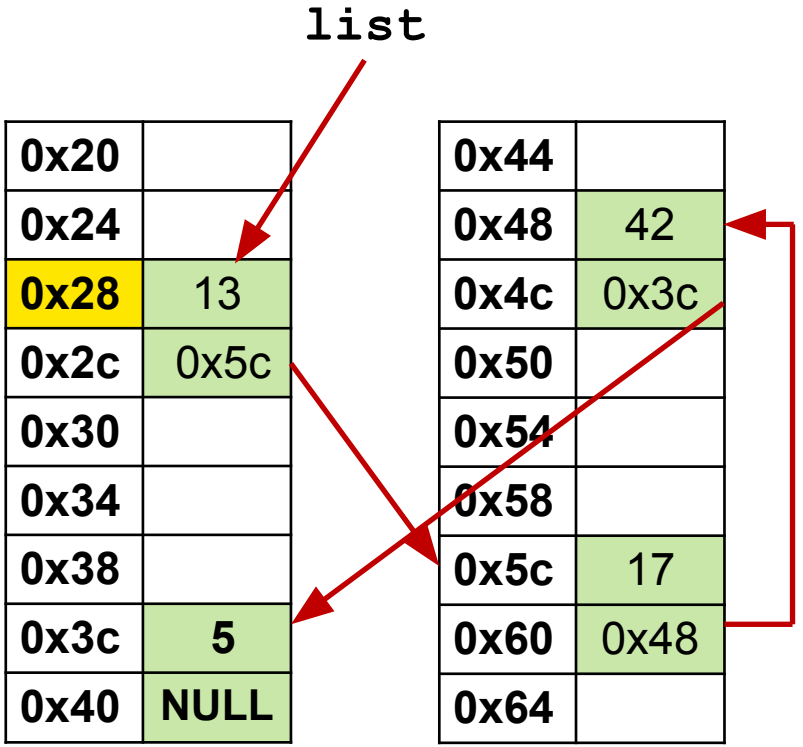
- We say it is sequential as we have to start at the beginning of the list and traverse to access items
- We can't jump to a particular item like we can with array indexes



Linked List Nodes

What type in C would allow us to store both the

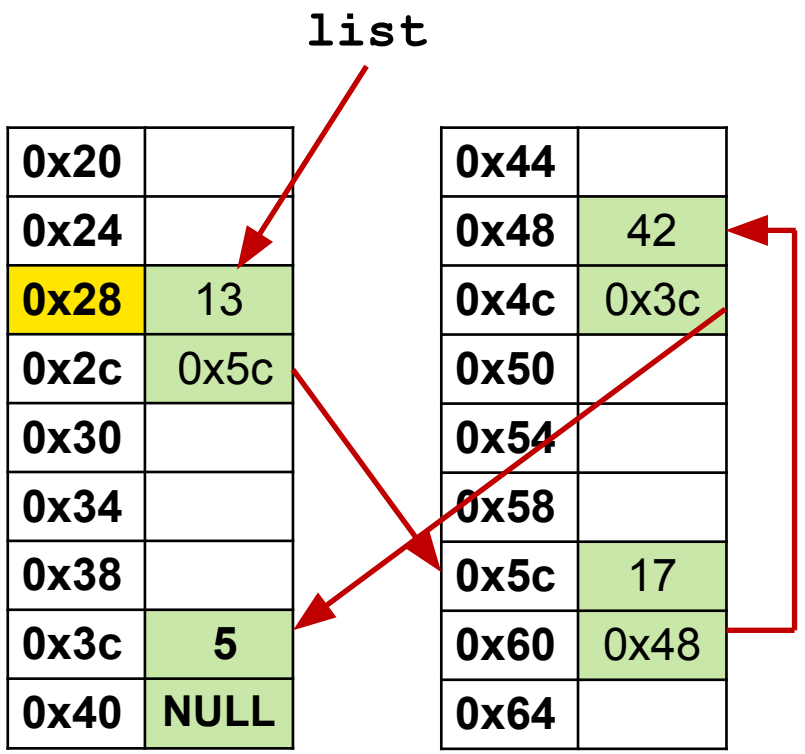
- **int** data and also the
- **address** of the next item in the list?



Linked List Nodes

- We can store our data and a pointer together in a struct.
- We often call these nodes when working with linked lists

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

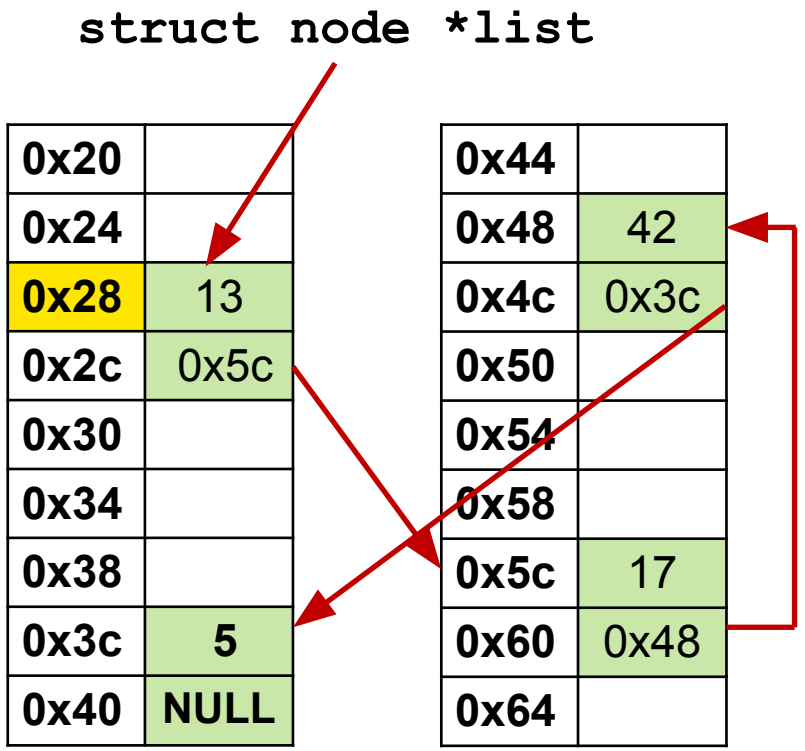


Linked List Nodes

The list variable is a pointer to the first node in the list

```
struct node *list;
```

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



Linked List Nodes

The list variable is a pointer to the first node in the list

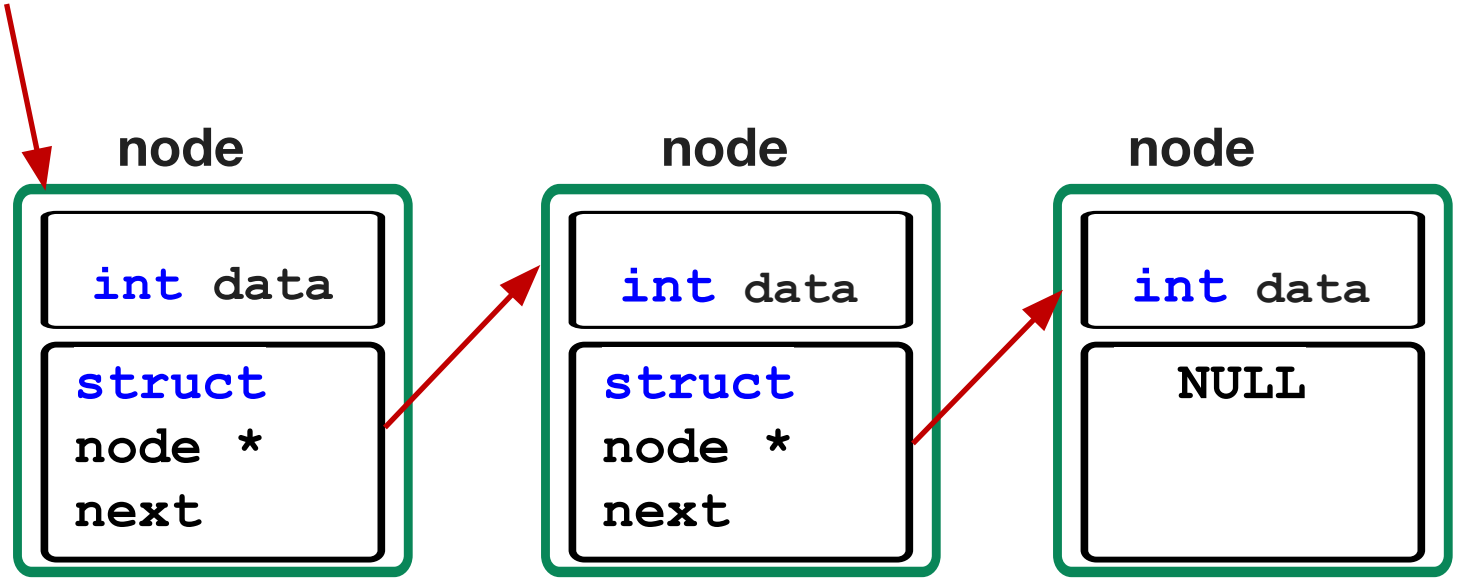
```
struct node *list;
```

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

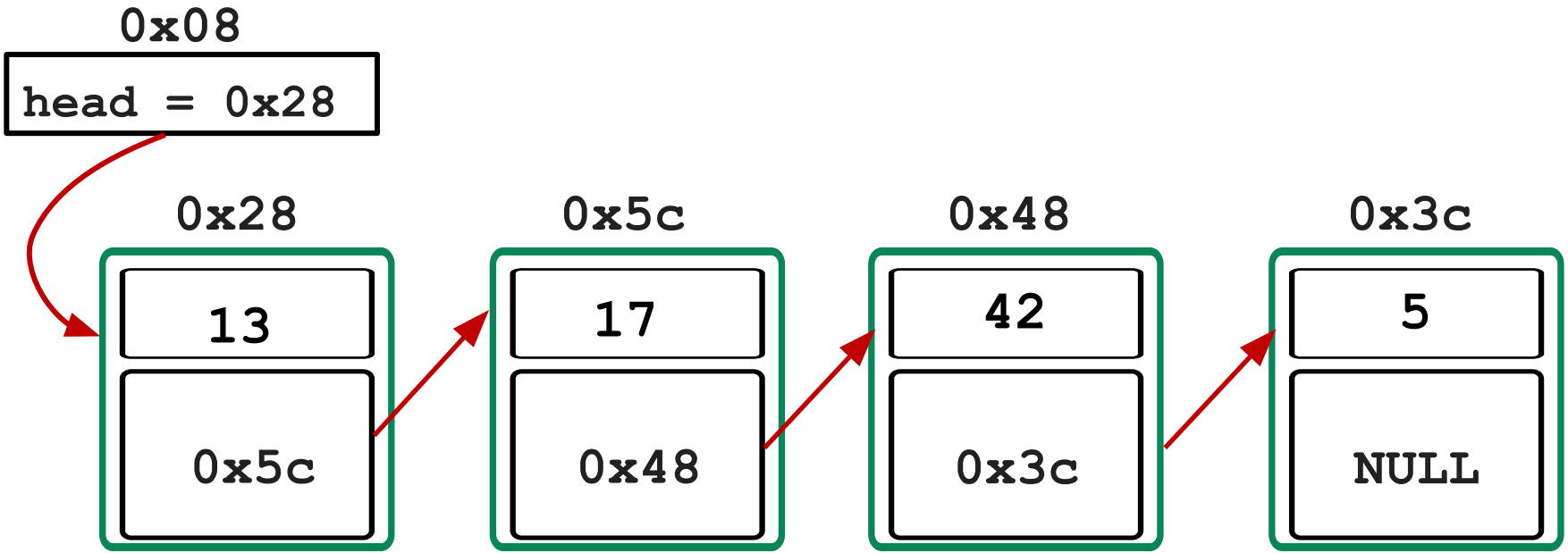
- Each node has some **data**
  - In this case it is one int but it could be whatever type of data you need
  - Later we will see different types of data in our linked lists
- Each node has a pointer to the **next** node (of the same data type)

Visualising Linked Lists

pointer to the first node in the list
(we often use the variable name head instead of list)



Visualising Linked Lists

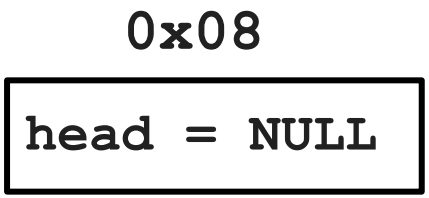


Creating a linked list

Let’s write the code to create a linked list with nothing in it.

```
struct node *head = NULL;
```

We can visualise it as follows



Hooray! Who said linked lists were difficult?

Creating a Node

We will be using **malloc** to create nodes on the **heap**.

- we want full control to be able to
    - create new nodes whenever we need to
    - free them whenever we are finished with them

Steps needed are:

1. malloc a struct node
2. set the data member in the node
3. set the pointer to the next node

Creating a List with 1 Node in C

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

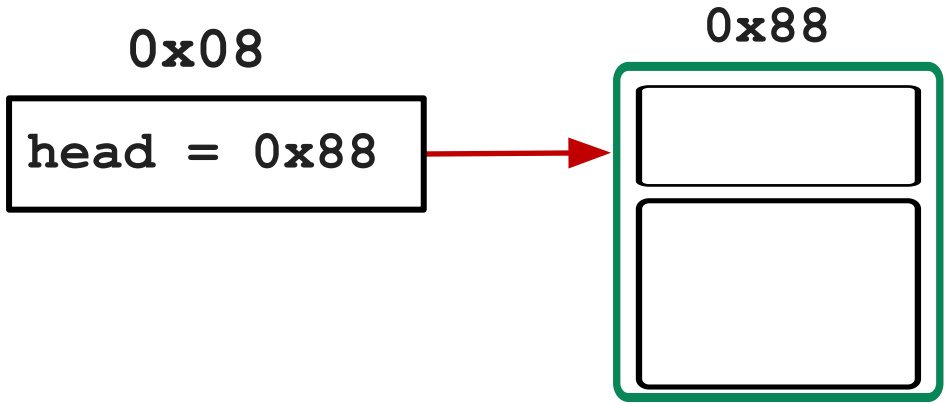
0x08

head = NULL

```
struct node *head = NULL;
```

Creating a List with 1 Node in C

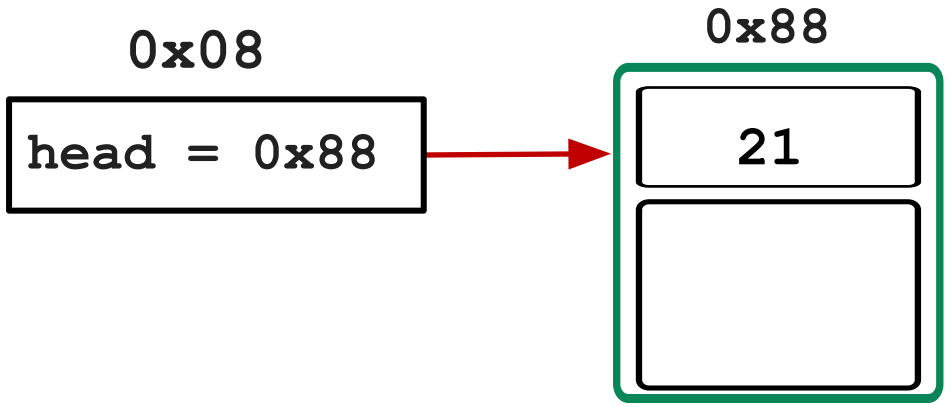
```
struct node {
    int data;
    struct node *next;
};
```




```
struct node *head = NULL;
head = malloc(sizeof(struct node));
```

Creating a List with 1 Node in C

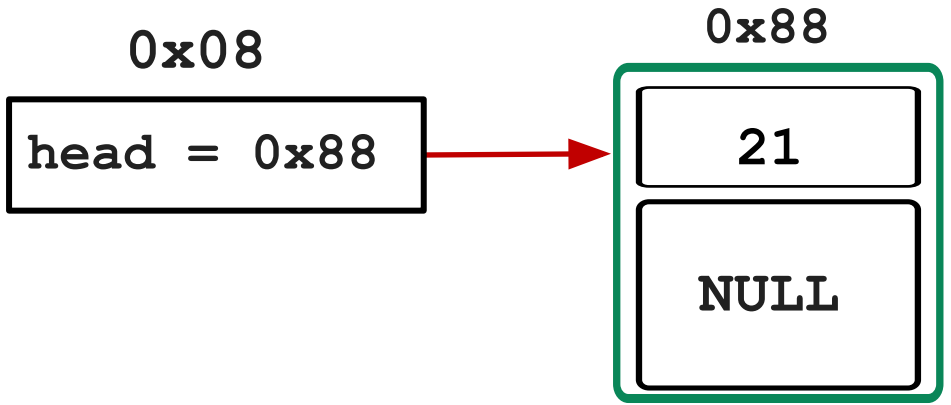
```
struct node {
    int data;
    struct node *next;
};
```




```
struct node *head = NULL;
head = malloc(sizeof(struct node));
head->data = 21;
```

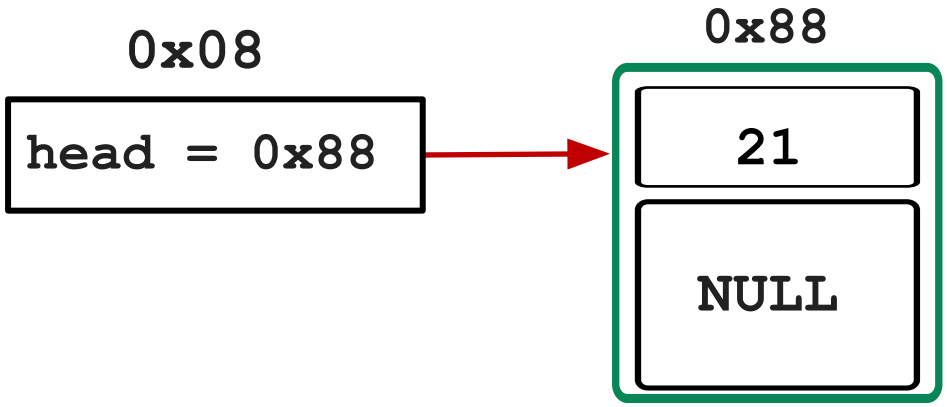
Creating a List with 1 Node in C

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




```c
struct node *head = NULL;
head = malloc(sizeof(struct node));
head->data = 21;
head->next = NULL;
```

Creating a List with 1 Node in C

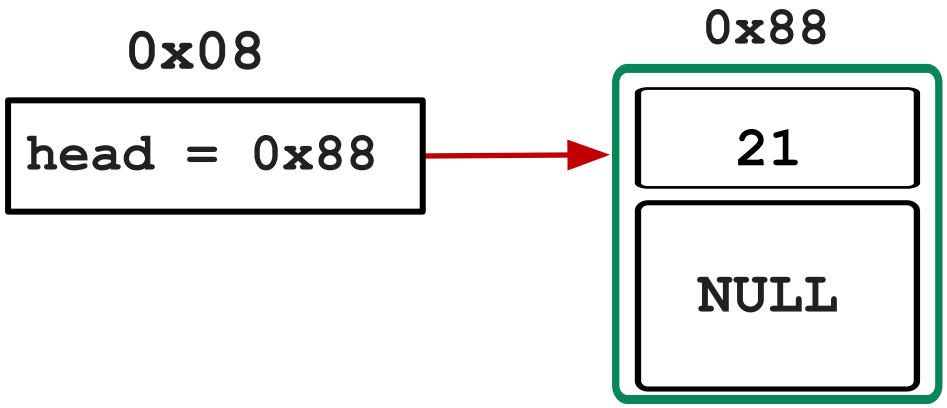


Now we have a linked list of size 1.

Let's create another node.

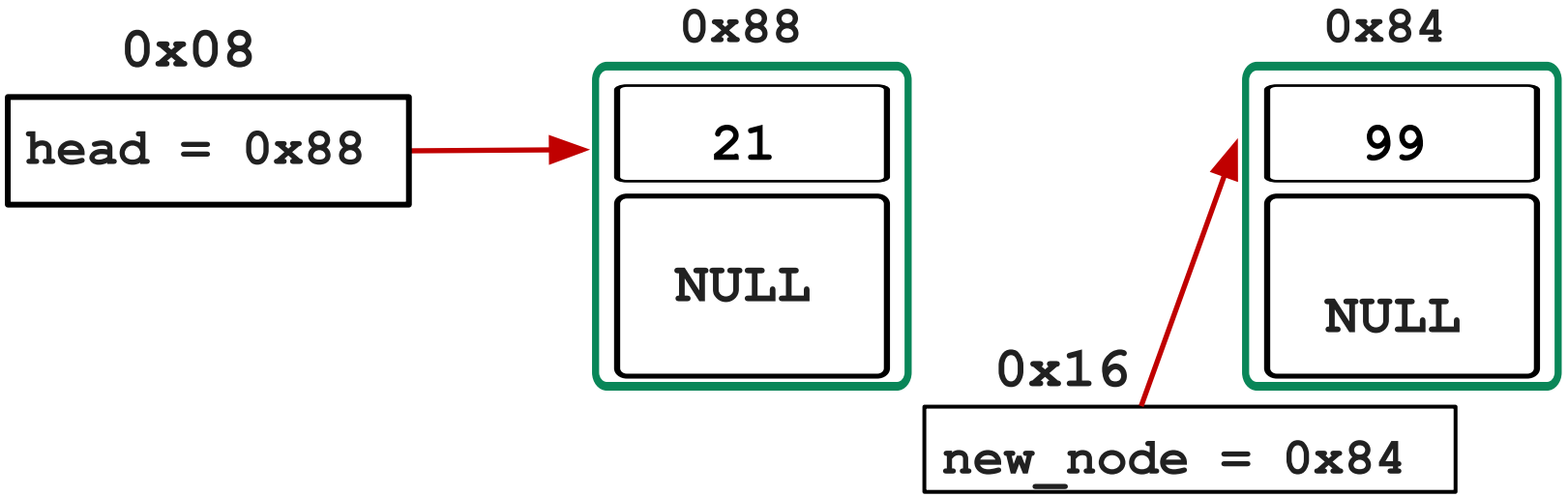
Then we can connect it to the end or the beginning of this list!

Connect 2 nodes: Add new node to the end



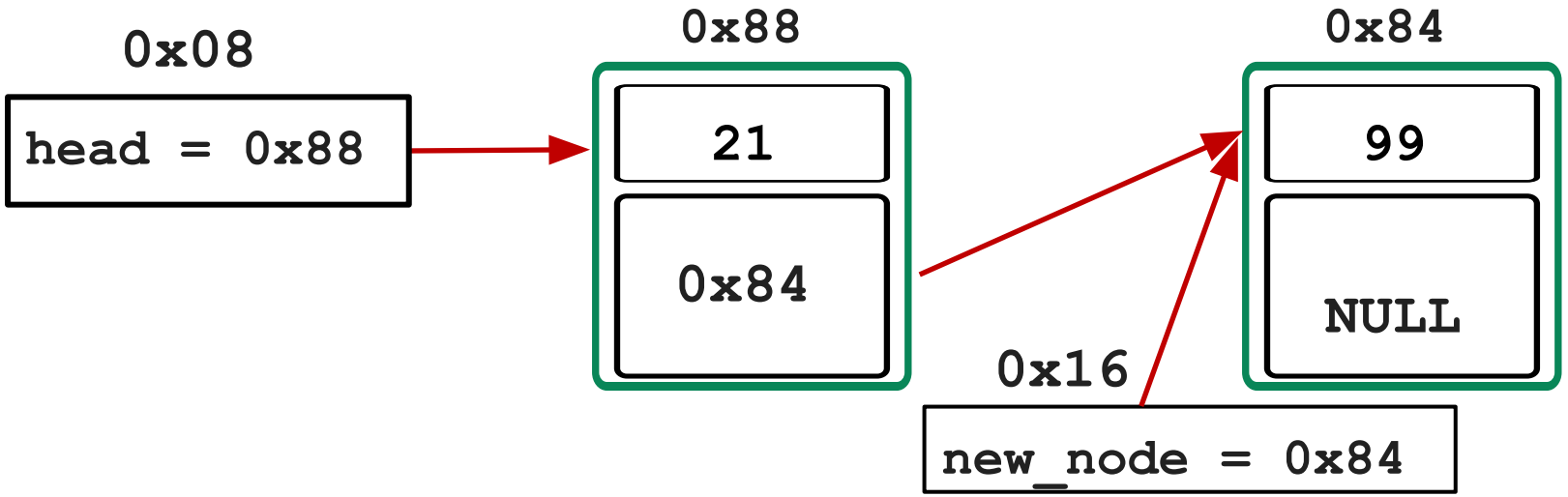
We will create a new node and link it to the end of this list
The end of the list is often called the tail.

Connect 2 nodes: Add new node to the end



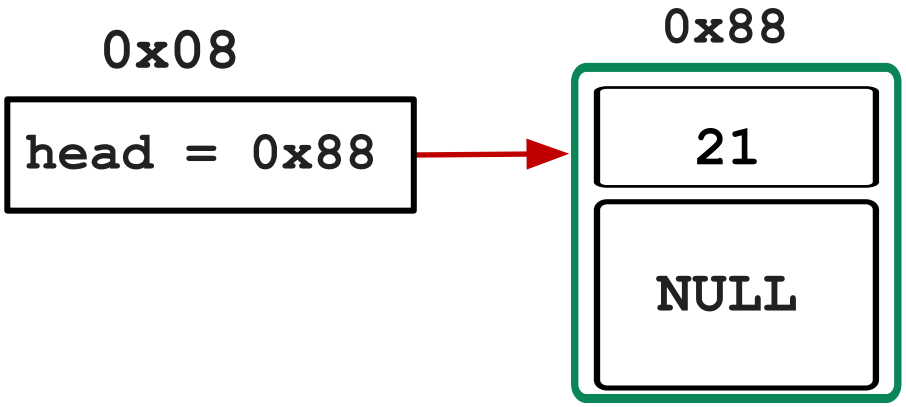

```
struct node *new_node = malloc(sizeof(struct node));
new_node->data = 99;
new_node->next = NULL;
```

Connect 2 nodes: Add new node to the end



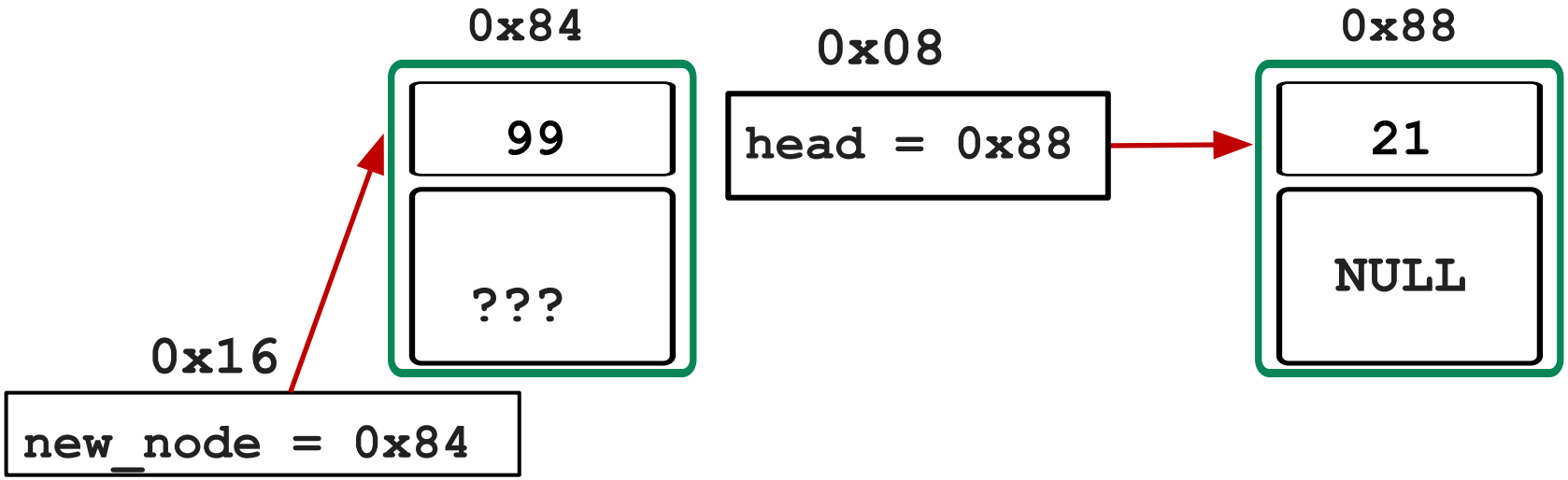

```
// Connect(link) the head of the list to the new_node
head->next = new_node;
```

Connect 2 nodes: Add new node to the start



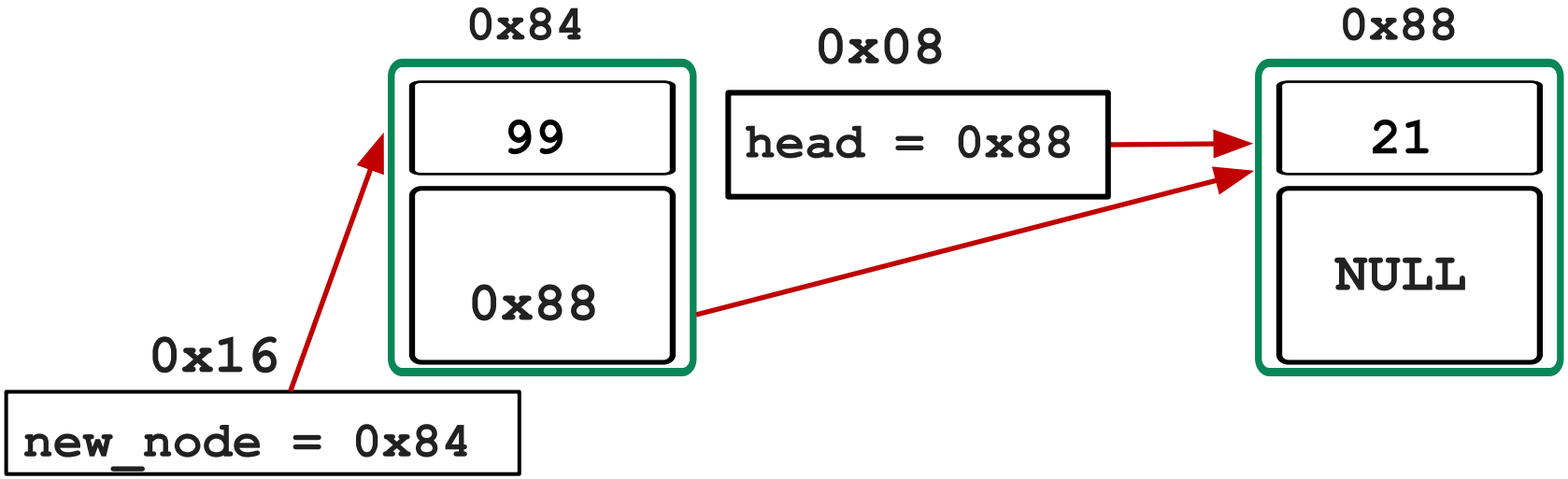
We will create a new node and link it to the start of this list
The start of the list is often called the head.

Connect 2 nodes: Add new node to the start



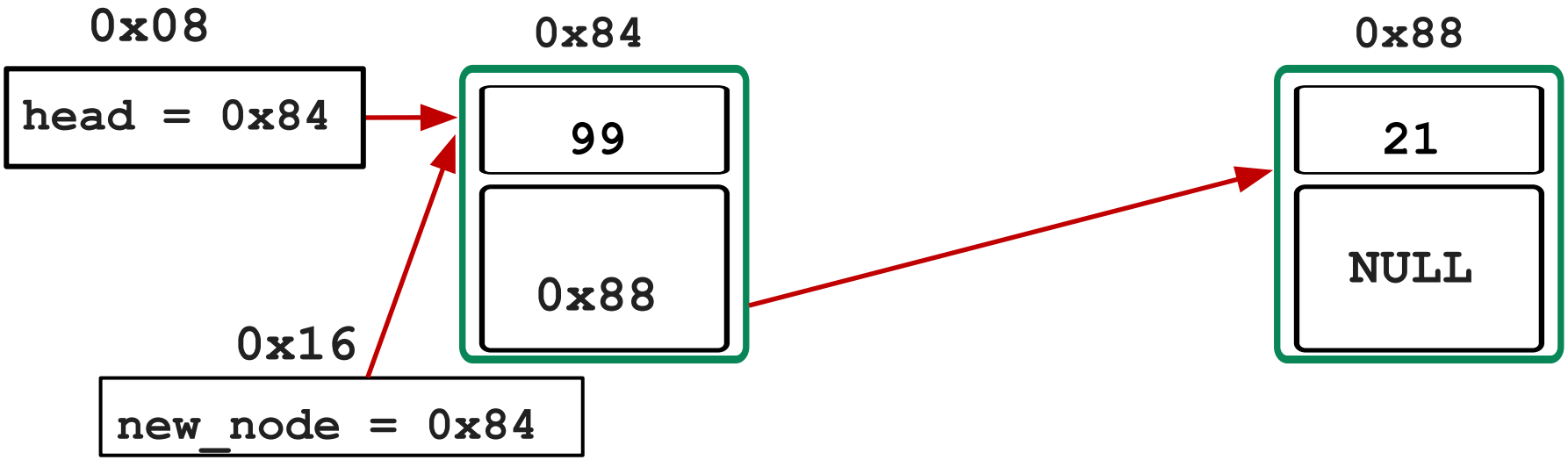

```
struct node *new_node = malloc(sizeof(struct node));
new_node->data = 99;
new_node->next = ???;
```

Connect 2 nodes: Add new node to the start




```
struct node *new_node = malloc(sizeof(struct node));
new_node->data = 99;
new_node->next = head;
```

Connect 2 nodes: Add new node to the start




```
head = new_node;
```

Coding Time

`linked_list_intro.c`

Create a list with 3 nodes

Print the contents of the first 3 nodes in the list

Code: Linked List Functions

`list_list_functions.c`

- How can we put our code to create a new node into a function?
- How could we use that to create a list by adding each node to head using a loop?
- How would we print the whole list? Even if it had 1000s of nodes?
- How could we add nodes to the end of the list? Even if it had 1000s of nodes?
- We want a function to free all nodes too. But let's leave that until another lecture…

Create Node Function

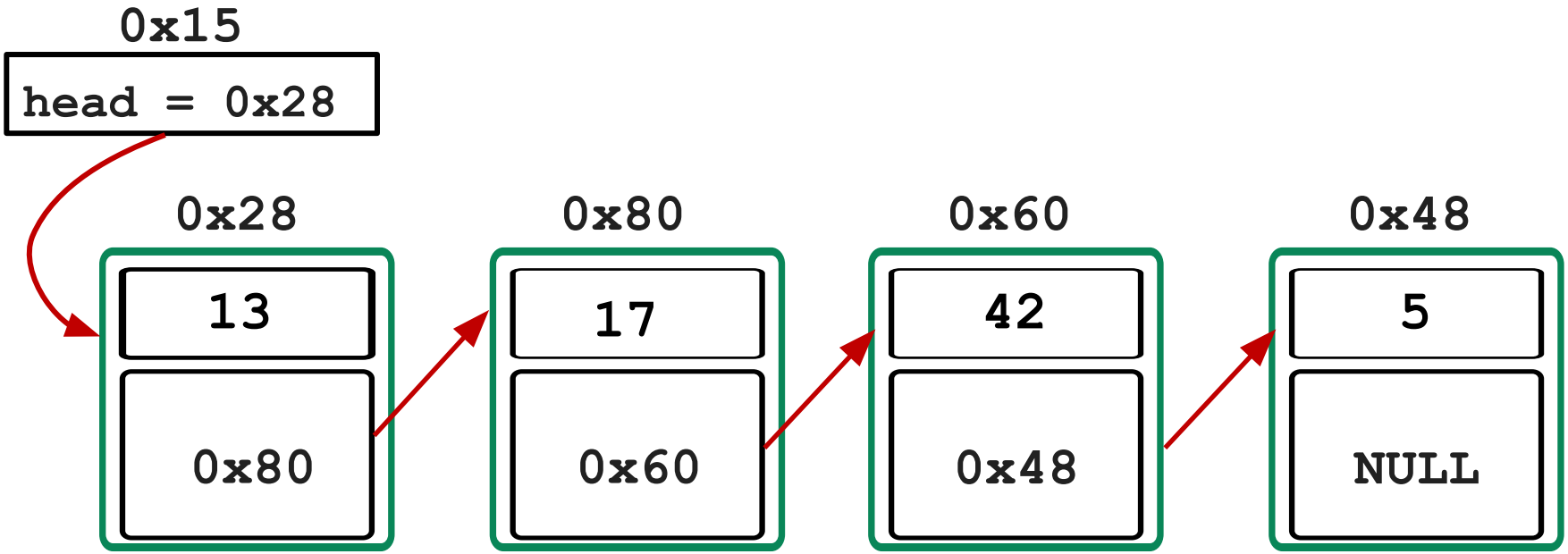
```
// Creates and returns a new node with given data and
// next pointer. returns NULL if memory allocation fails.
struct node *create_node(int data, struct node *next){
    struct node *new_node = malloc(sizeof(struct node));
    if (new_node == NULL) {
        return NULL;
    }
    new_node->data = data;
    new_node->next = next;
    return new_node;
}
```

Creating a Linked List Inserting at Head

```
// What would the contents of our list be?
int main(void) {
    struct node *head = NULL;
    for(int i = 0; i < 10; i++) {
        struct node *new_node = create_node(i, head);
        head = new_node;
    }
    return 0;
}
```

Printing a Node

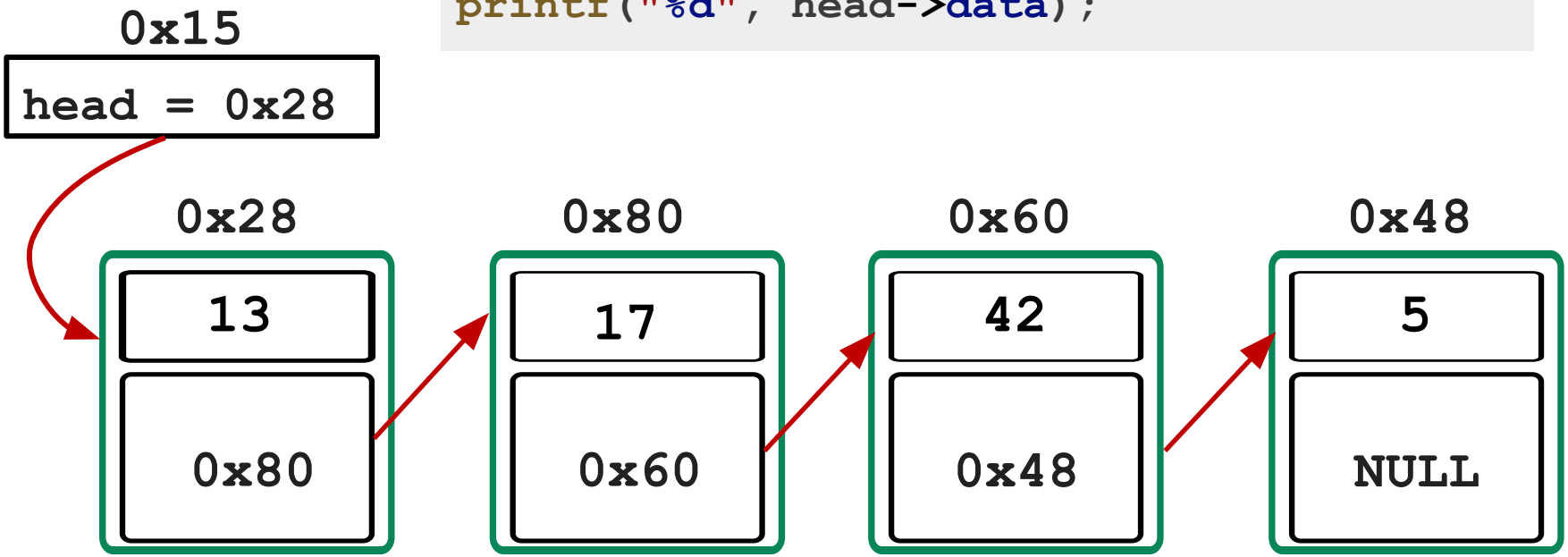
How could I print the data from the first node in this linked list?



Printing a Node

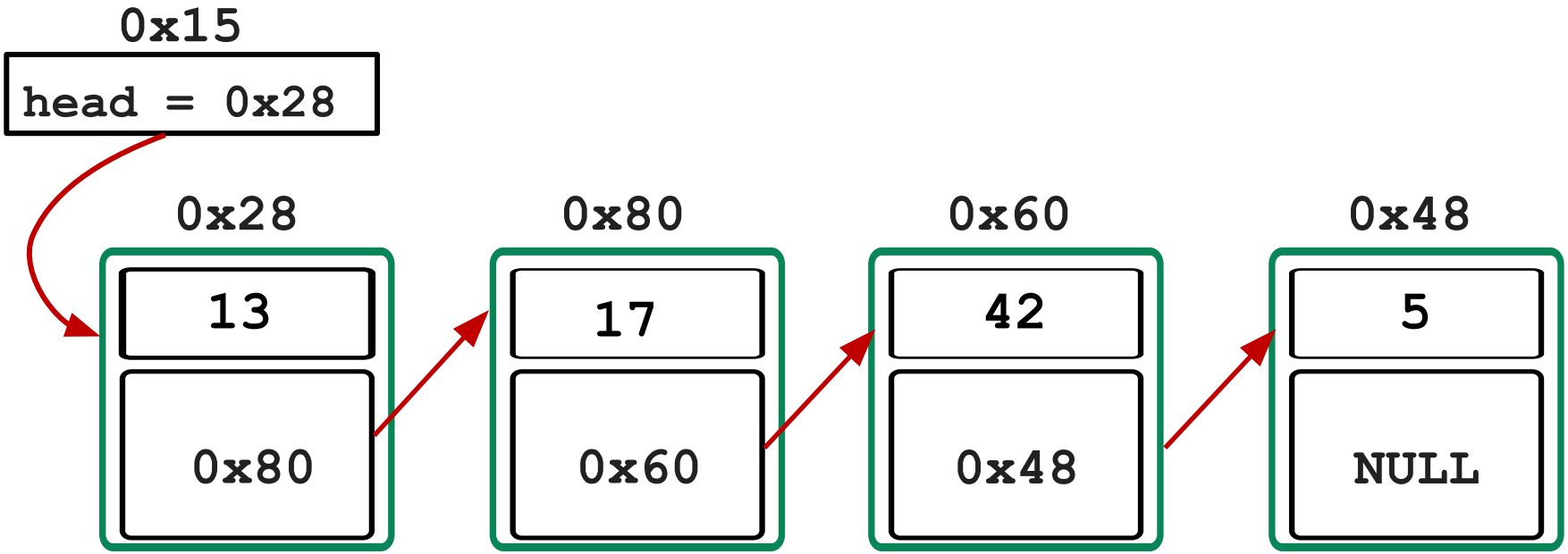
How could I print the data from the first node in this linked list?

```
printf("%d", head->data);
```



Printing a Linked Lists

How could I print data from **each** node in this linked list?



Traversing a list

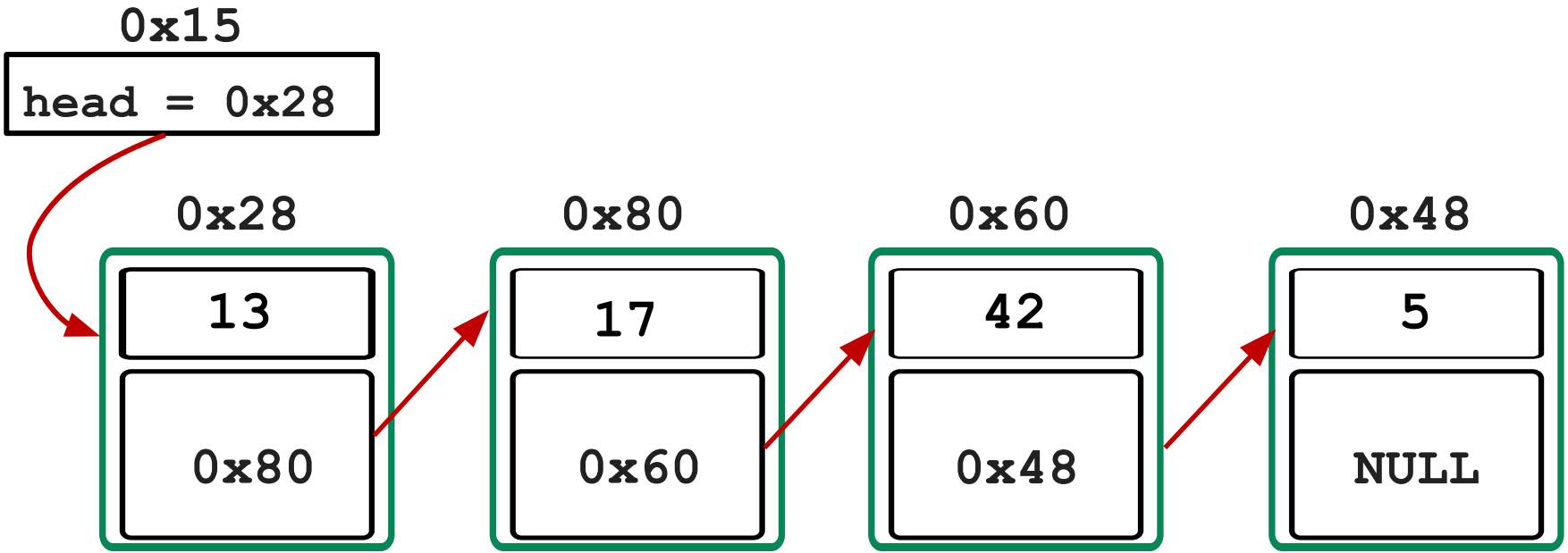
Traversing a list means

- starting at the head of the list
- moving node by node until we get to the end of the list.

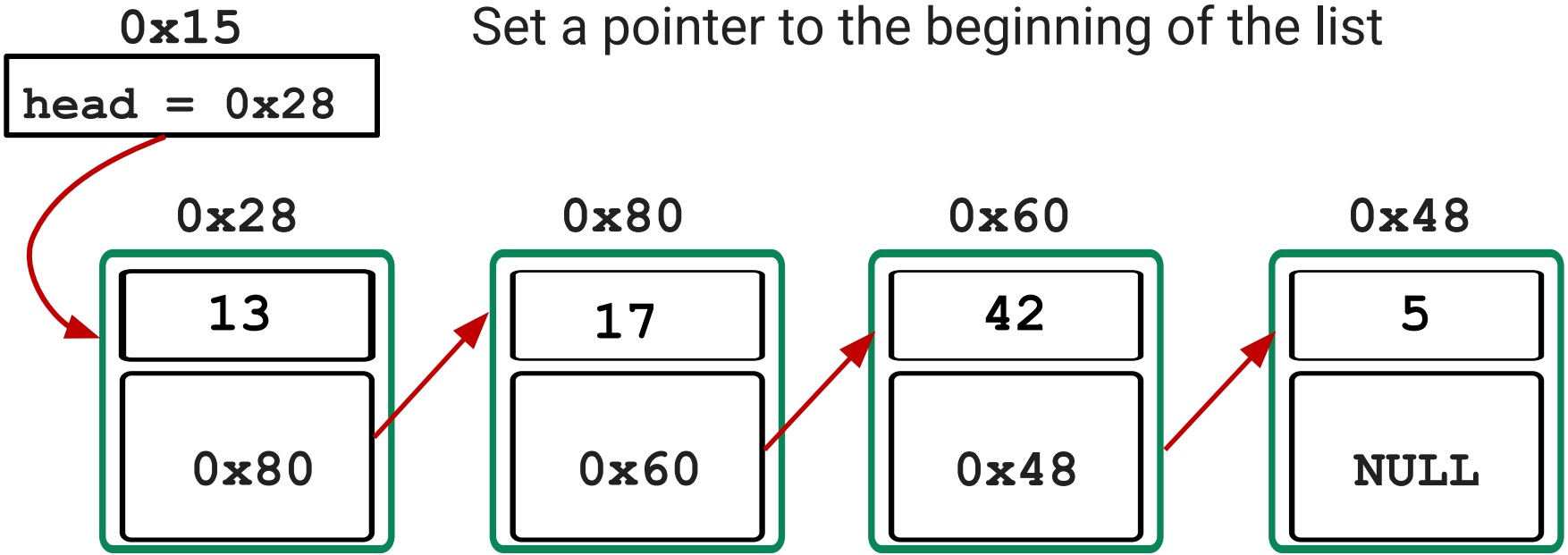
We often want to traverse a list, node by node to do things like

- print the data in each node in the list
- count the number of nodes in the list
- search for data in the list

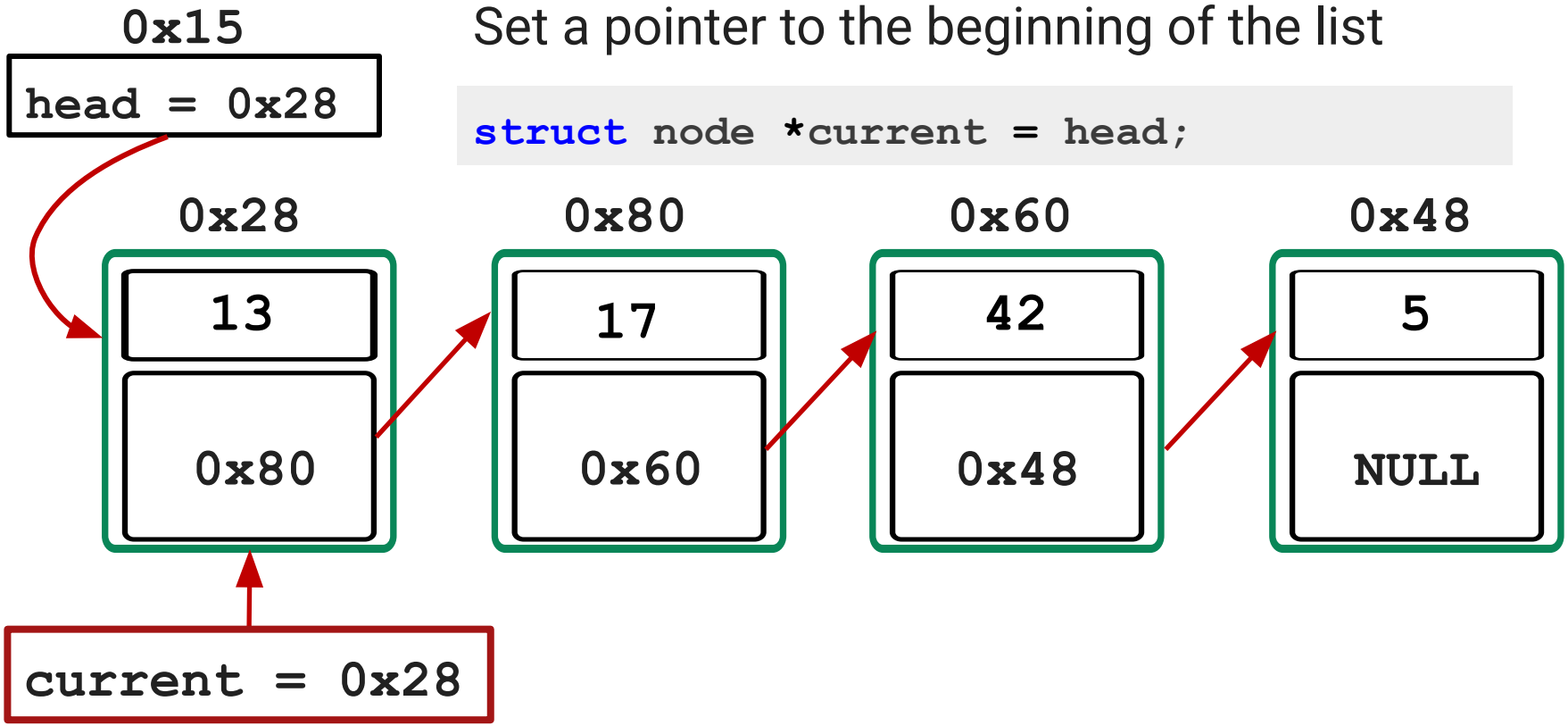
Traversing a List



Traversing a List



Traversing a List

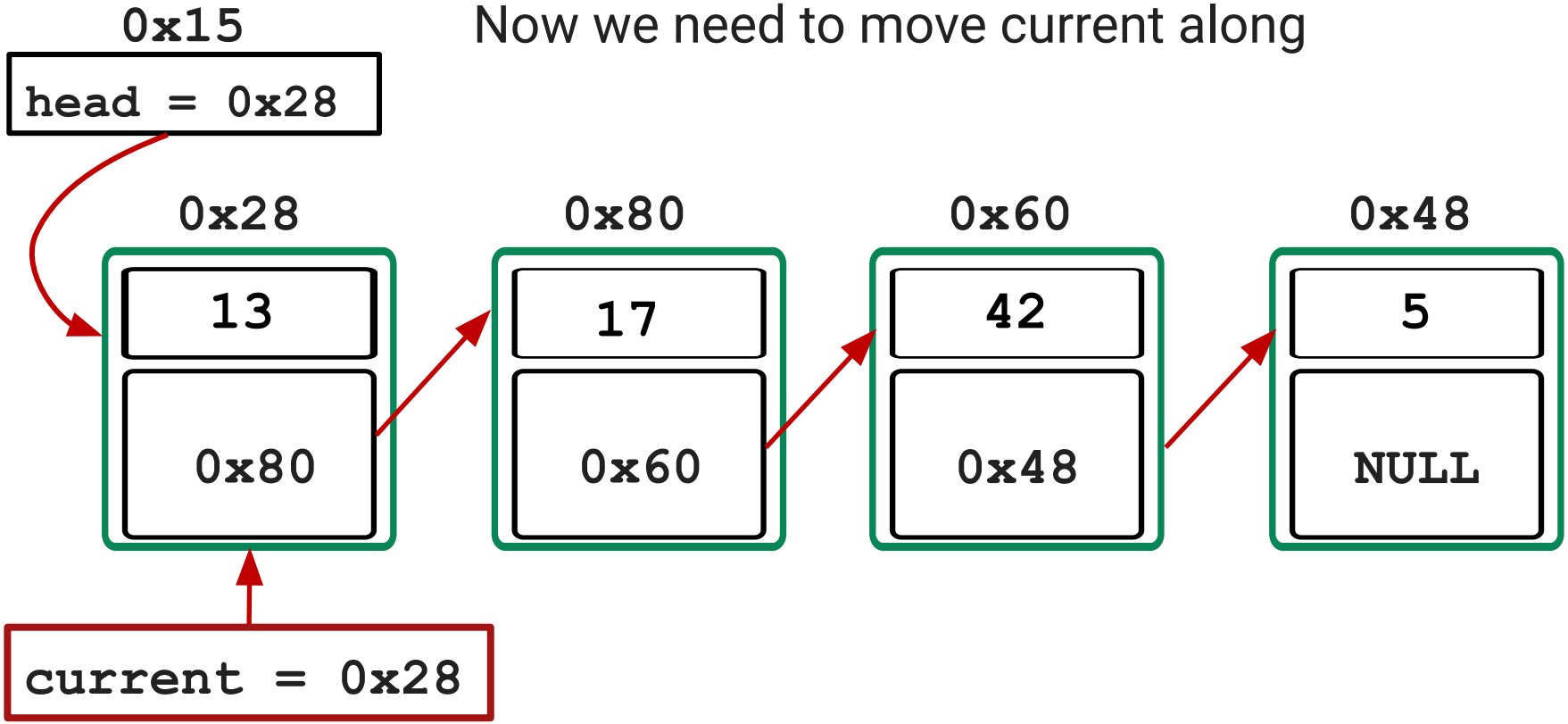


Set a pointer to the beginning of the list

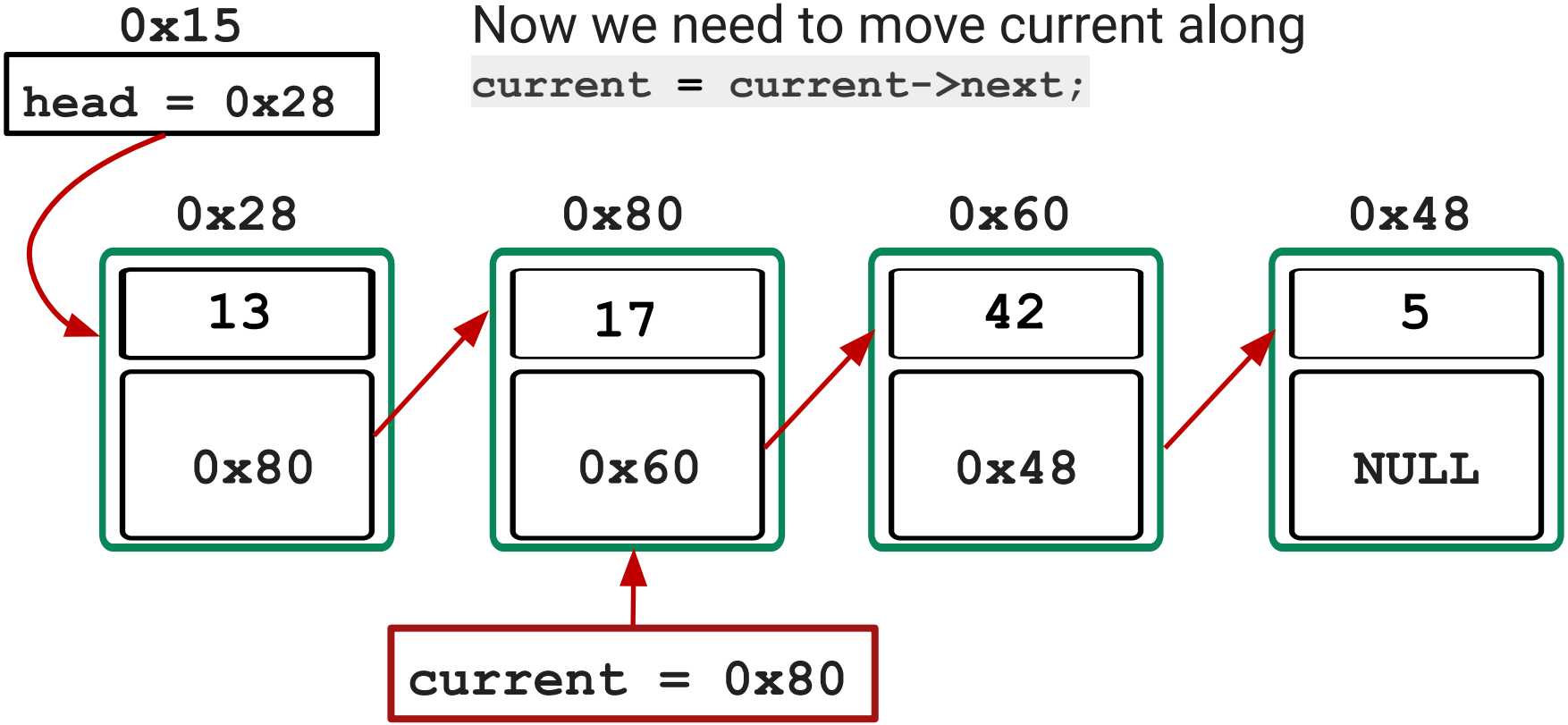
```
struct node *current = head;
```

Traversing a List

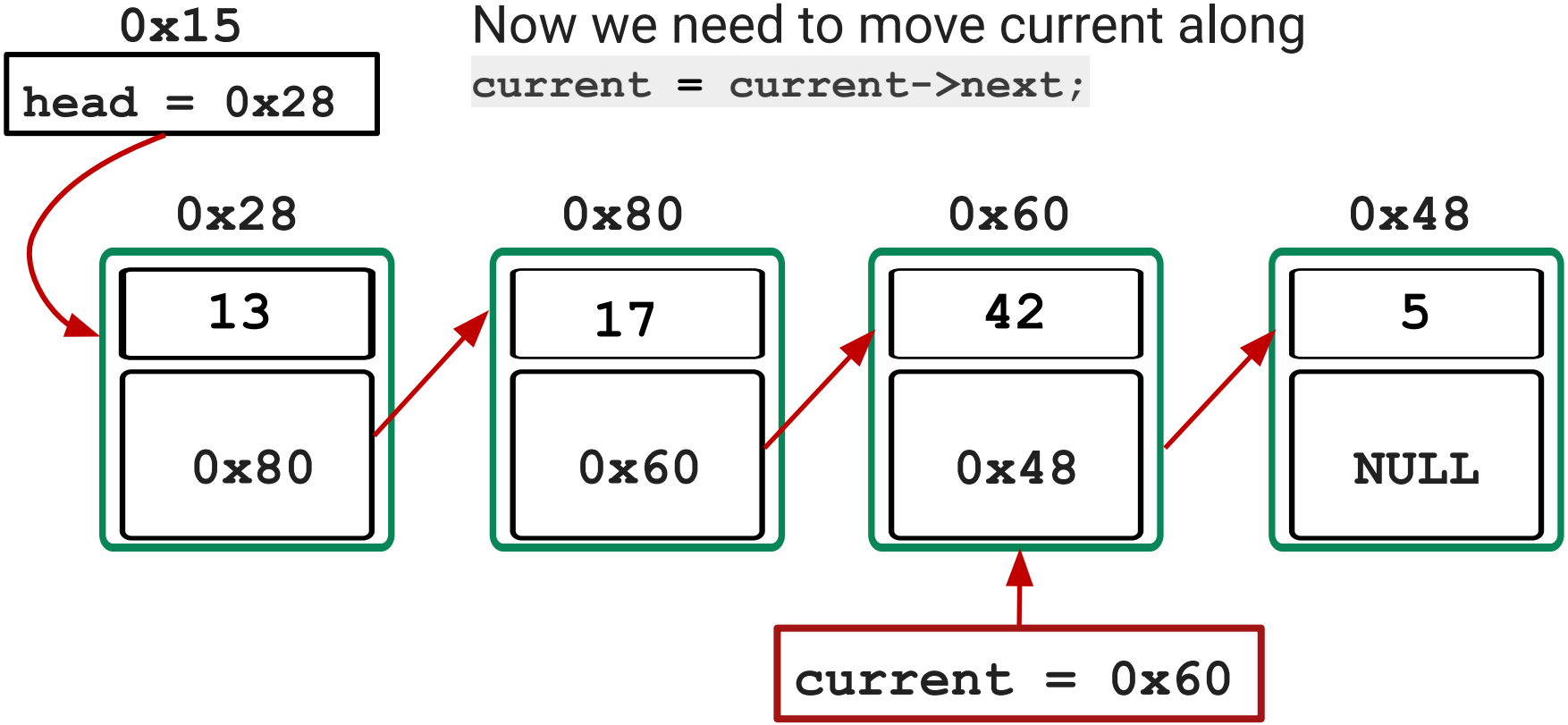
Now we need to move current along



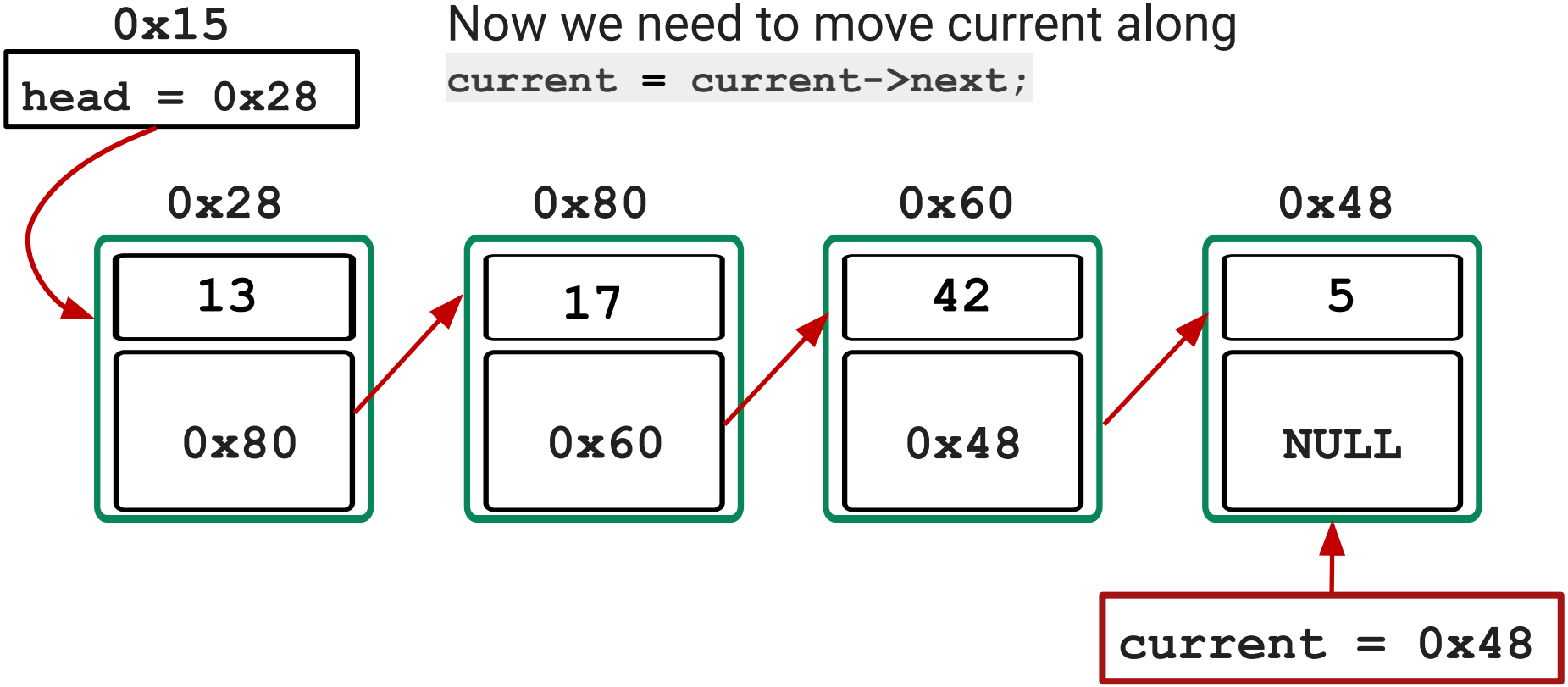
Traversing a List



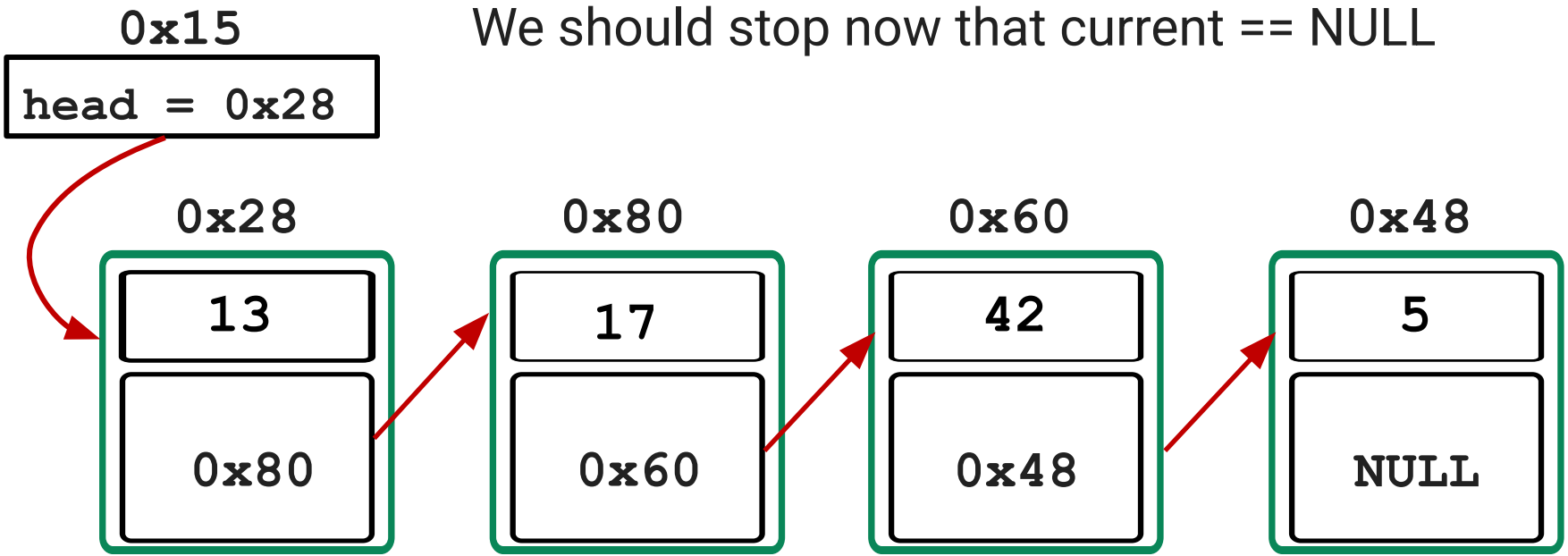
Traversing a List



Traversing a List



Traversing a List



Printing a list

```c
// Traversing the list and printing the contents (data)
// from each node
void print_list(struct node *head) {
    struct node *current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

Inserting nodes in a linked list

Where can I insert in a linked list?

- At the head (what we just did!)
- Between any two nodes that exist (next lecture!)
- After the tail as the last node (next lecture!)

What did we learn today?

- Realloc (realloc_array.c)
- Lists Intro (linked_list_intro.c)
- Inserting nodes at the start of the list (linked_list_functions.c)
- Traversing a List

Next lecture:

- Inserting an element anywhere in the list!
- Deleting an element
- Lists containing other types of data

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/nTz8Wkd0vB

Reach Out

Content Related Questions:

Forum

Admin related Questions email:

cs1511@unsw.edu.au