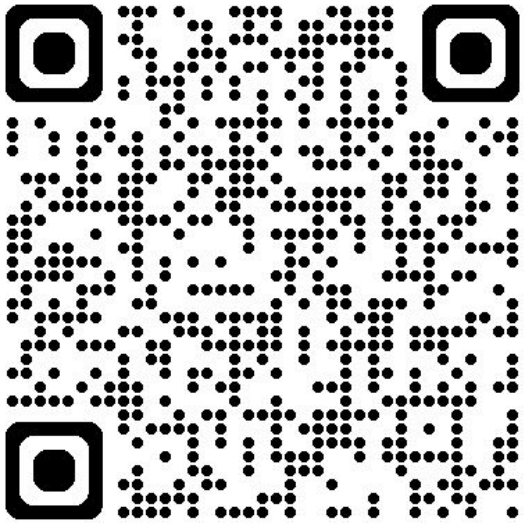
# COMP1511/1911 Programming Fundamentals

## Week 9 Lecture 1

# Linked Lists
# A larger Application

# Link to Week 9 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T3/code/week_9/



# Announcements

- **Assignment 1 Marks**:
  - Hopefully out later today: look at feedback not just mark
- **Revision Sessions:**
  - Last set of revision sessions on in week 11
  - Look out for announcement and sign ups on the forum soon

# Week 10 Practice Exams

- Held in Labs for 2 hours
- This is how you get lab marks for week 10
  - Marks are based on attempting it.
- If you are in an online tut-lab
  - you can sign up for an in-person lab for week 10
    https://buytickets.at/comp1511unsw/1857368
- Don’t miss another chance to see what the exam environment is like and get used to using it. And try out some linked list hurdles and other exam questions in an exam environment.

# Last Week

- Inserting Nodes anywhere
- Deleting Nodes
  - From the start of the list
  - Freeing all nodes
  - Search and Delete Approach 1

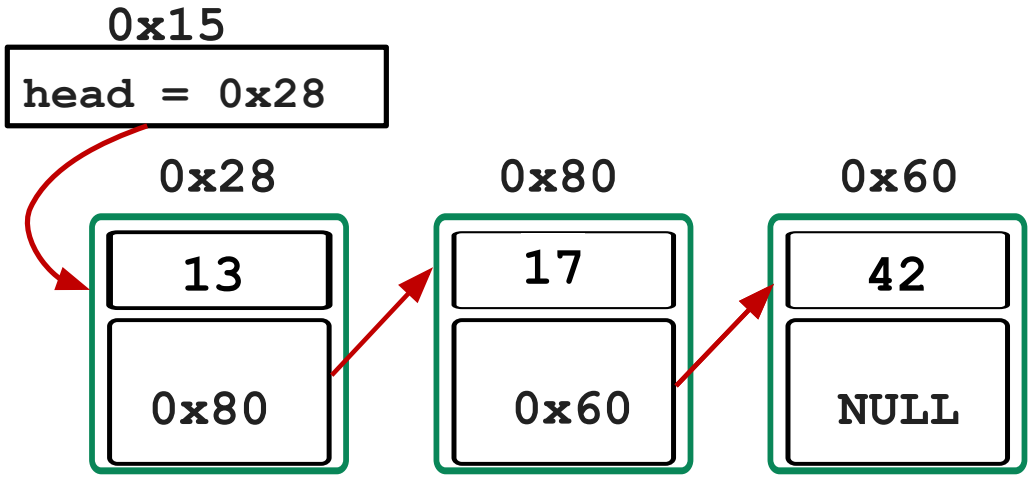
# Today's Lecture

- Recap Deletion:
  - Delete First Node
  - Free all nodes
  - Linked List Search and Delete Exercise
    - Second implementation debugging
    - Extending first implementation to delete all occurrences
- Linked Lists a Larger Application.
  - Linked Lists with complex data (other than just int)
  - Multi-file Linked Lists
  - Helpful for assignment 2

# Linked List Exercise

- How could I print the data in the first node?
- How could I print the data in the second node?
- How could I modify the next field in the first node to be NULL? What would that do?

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



# Deletion Recap

# Deleting the First Node in a Linked List

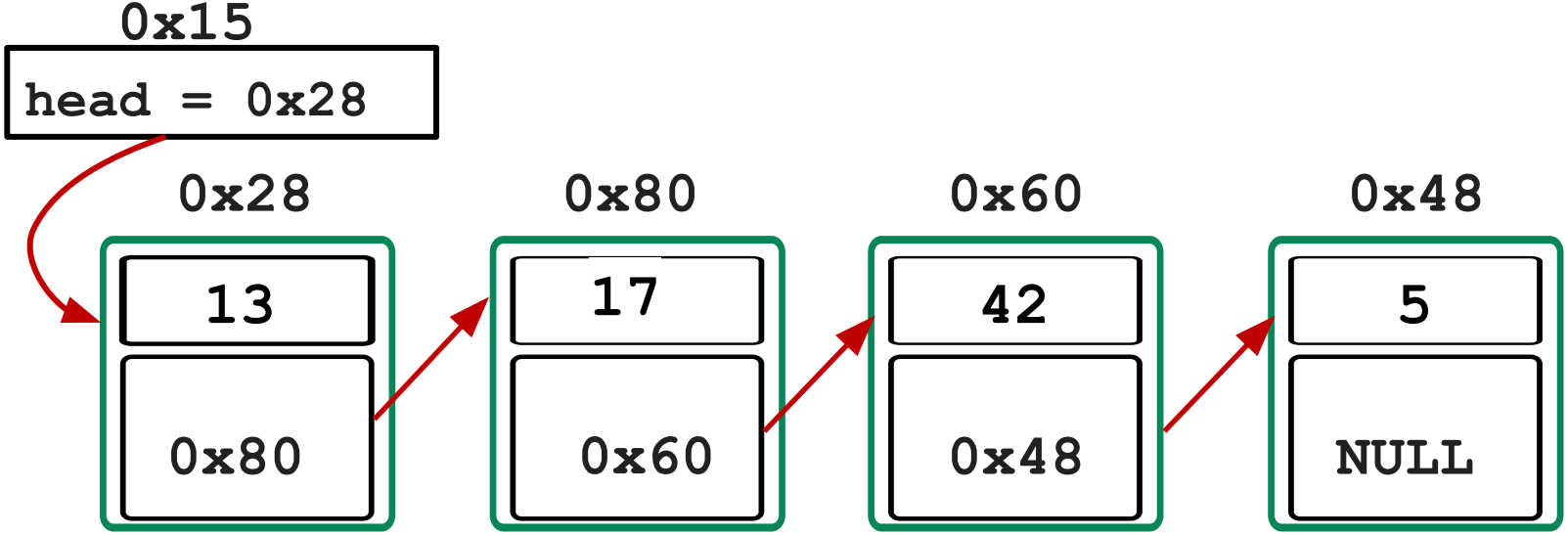
Let's write a function to delete the first node in a linked list.
We need to consider the case when the list is empty

- If it is empty we can't delete anything
- We just return the head of the list which would be  NULL

```
if (head == NULL) {
    return head; //or return NULL;
}
```

# Deleting the First Node in a Linked List

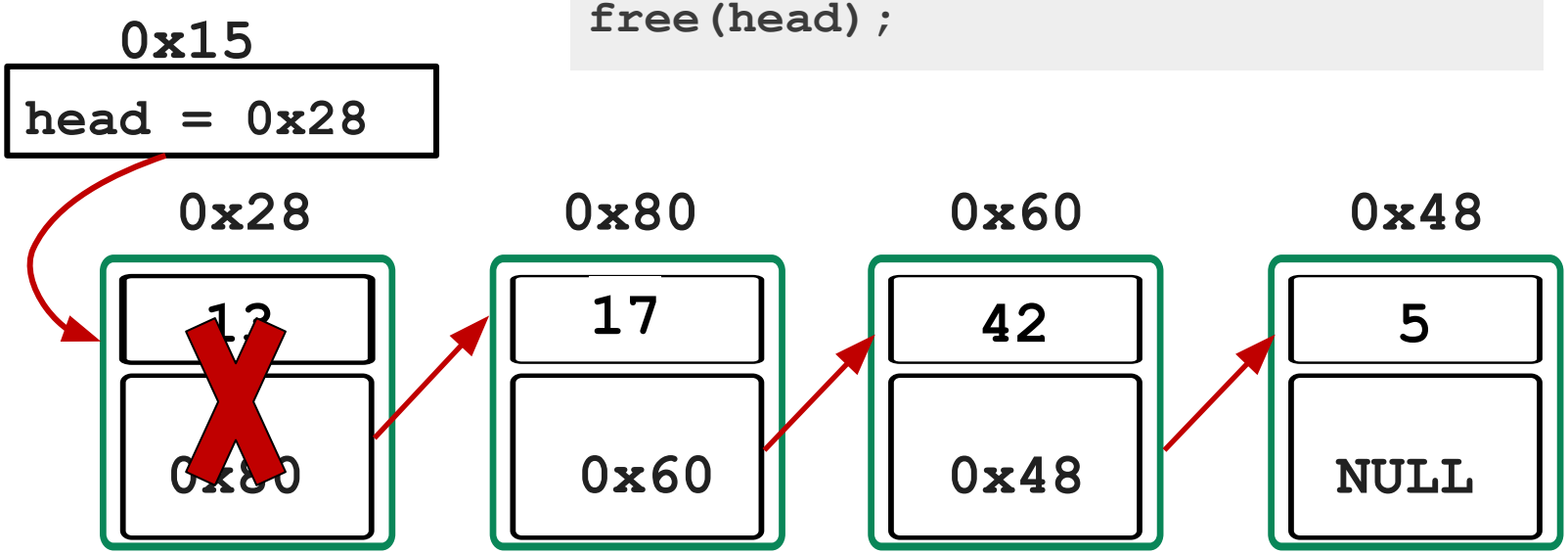
If our list is not empty, we want to make the second node the new head of the list and free the first node that we want to delete.



# Deleting the First Node in a Linked List

What would be the problem calling free on head first?

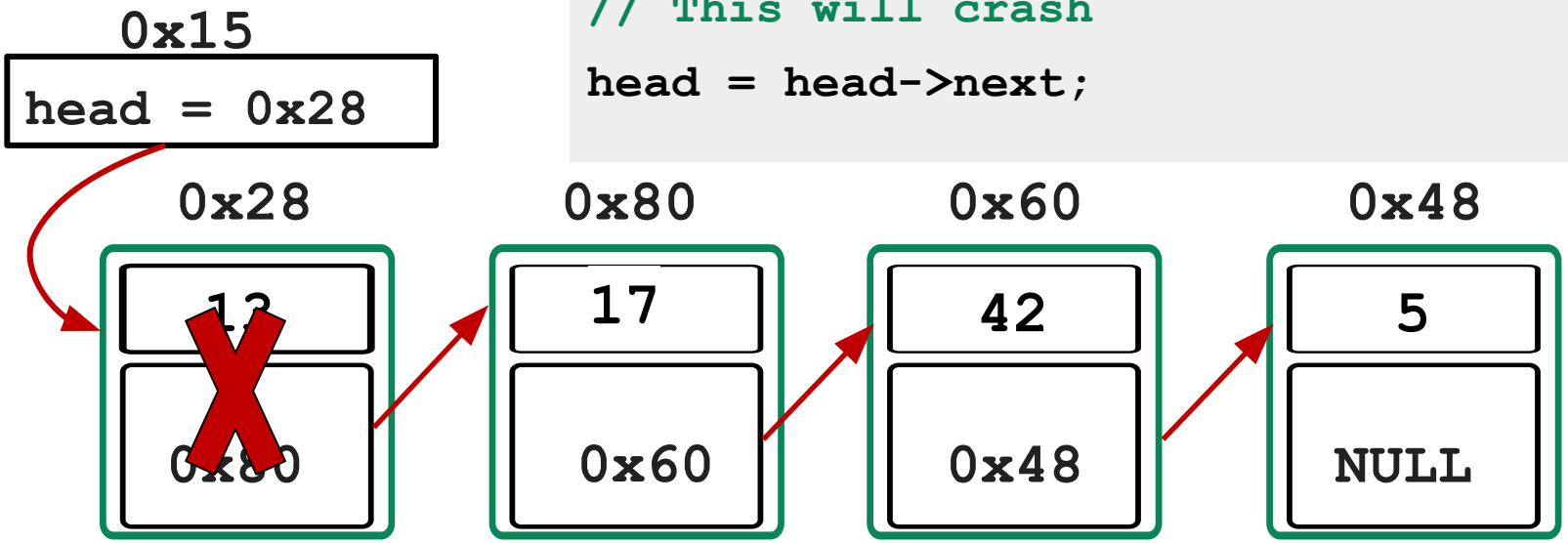
```
free(head);
```



# Deleting the First Node in a Linked List

We can't access memory that has been freed. We have lost the rest of the list

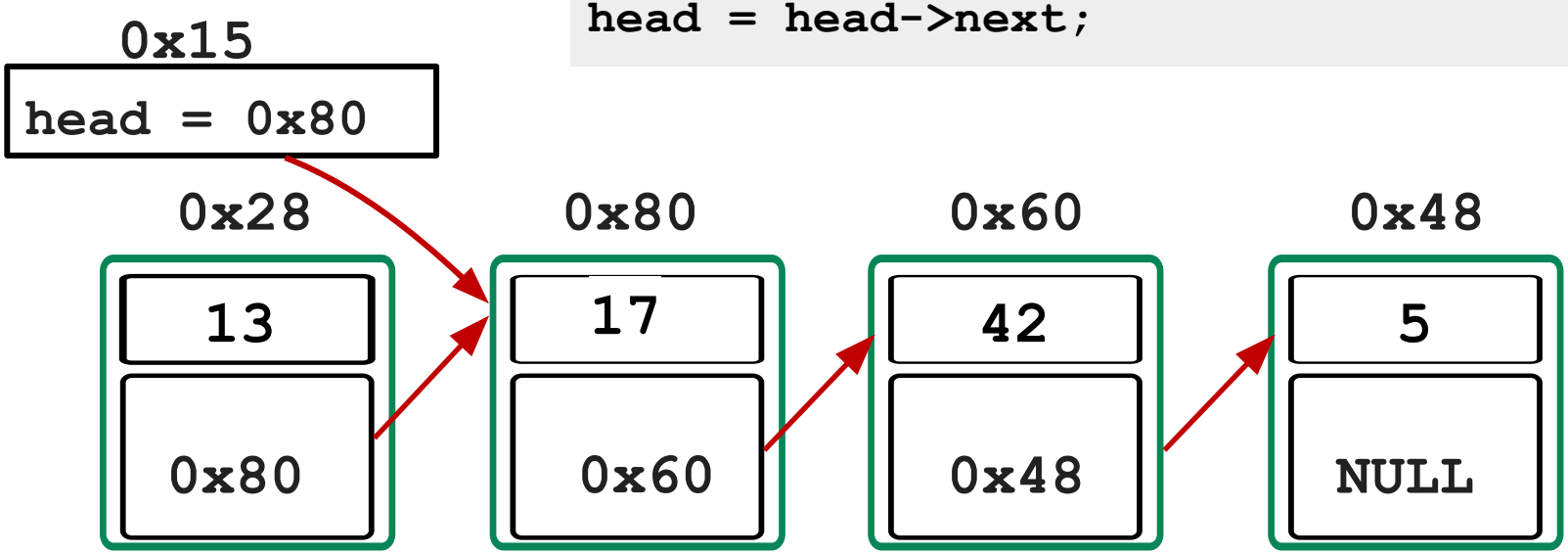
```
// This will crash
head = head->next;
```



# Deleting the First Node in a Linked List

What would be the problem with updating head first?

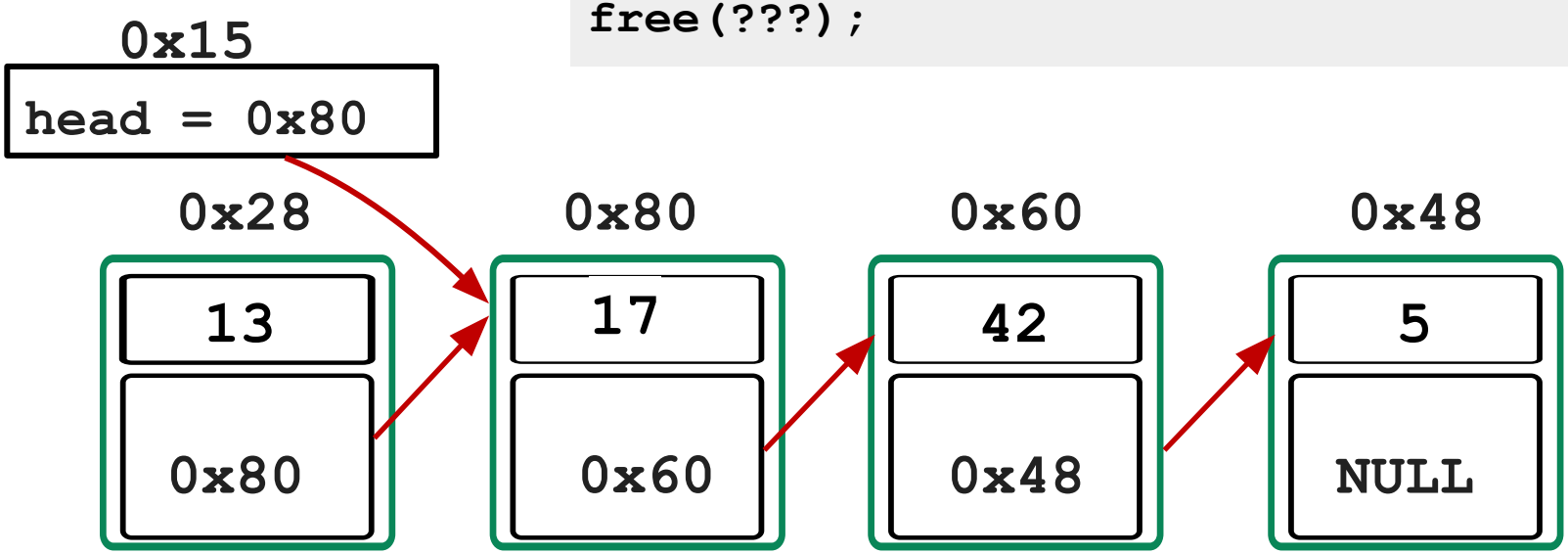
```
head = head->next;
```



# Deleting the First Node in a Linked List

We now have no pointer to the first node so we can't free it!

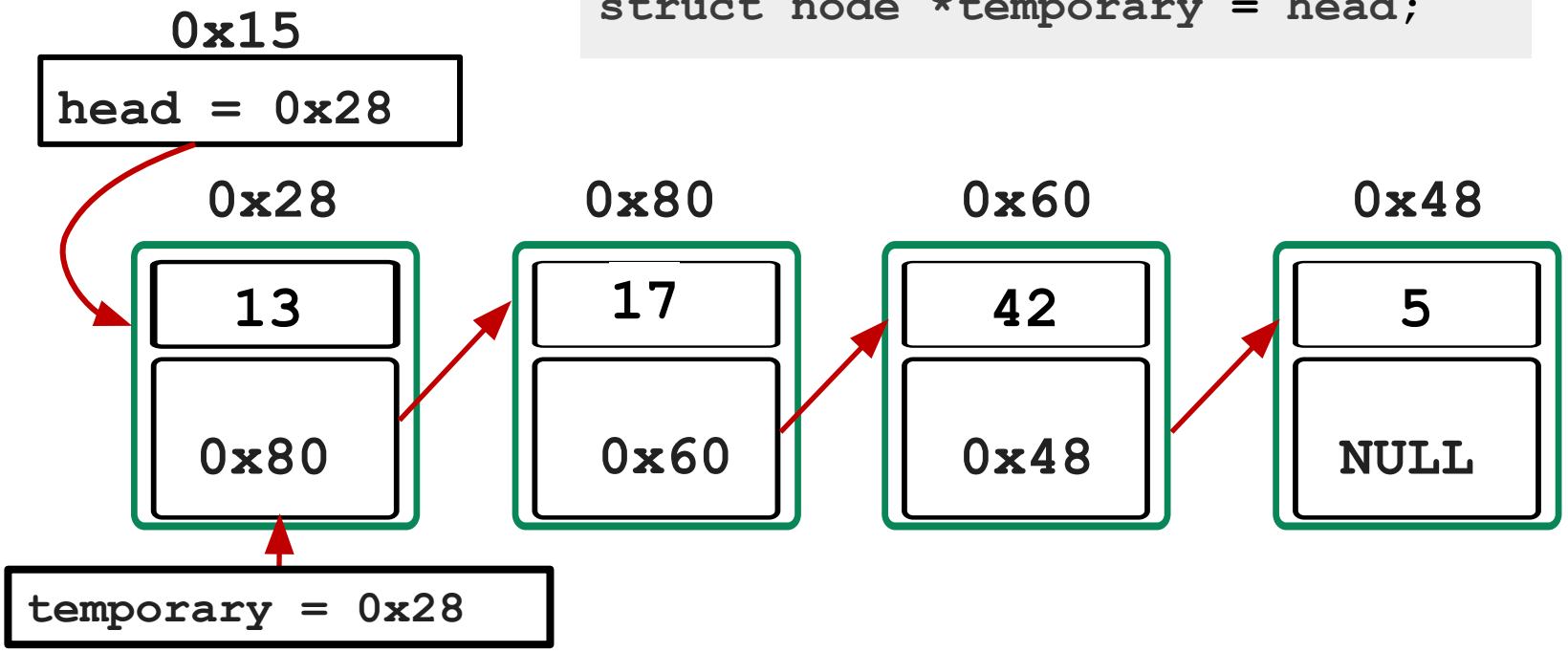
```
free(???);
```



# Deleting the First Node in a Linked List

Let's create a pointer to the first node

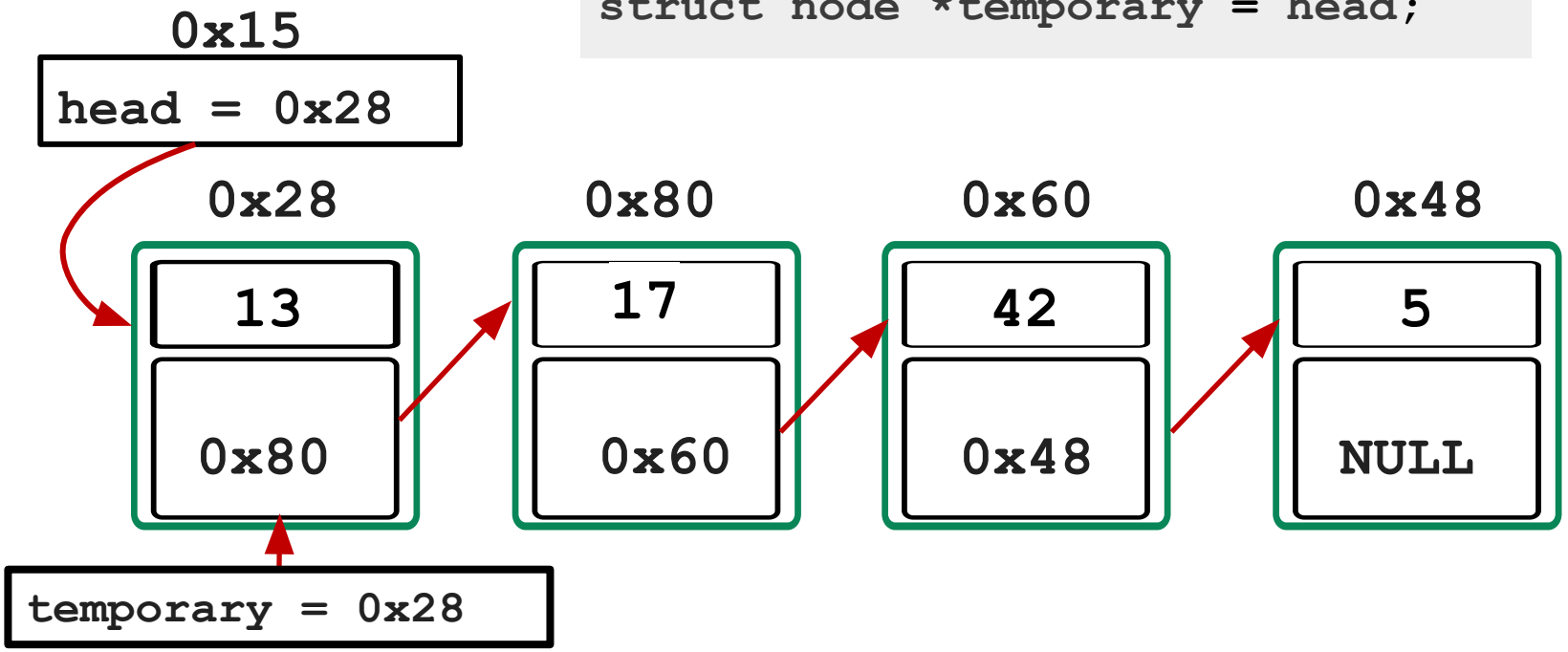
```
struct node *temporary = head;
```



# Deleting the First Node in a Linked List

Let's create a pointer to the first node

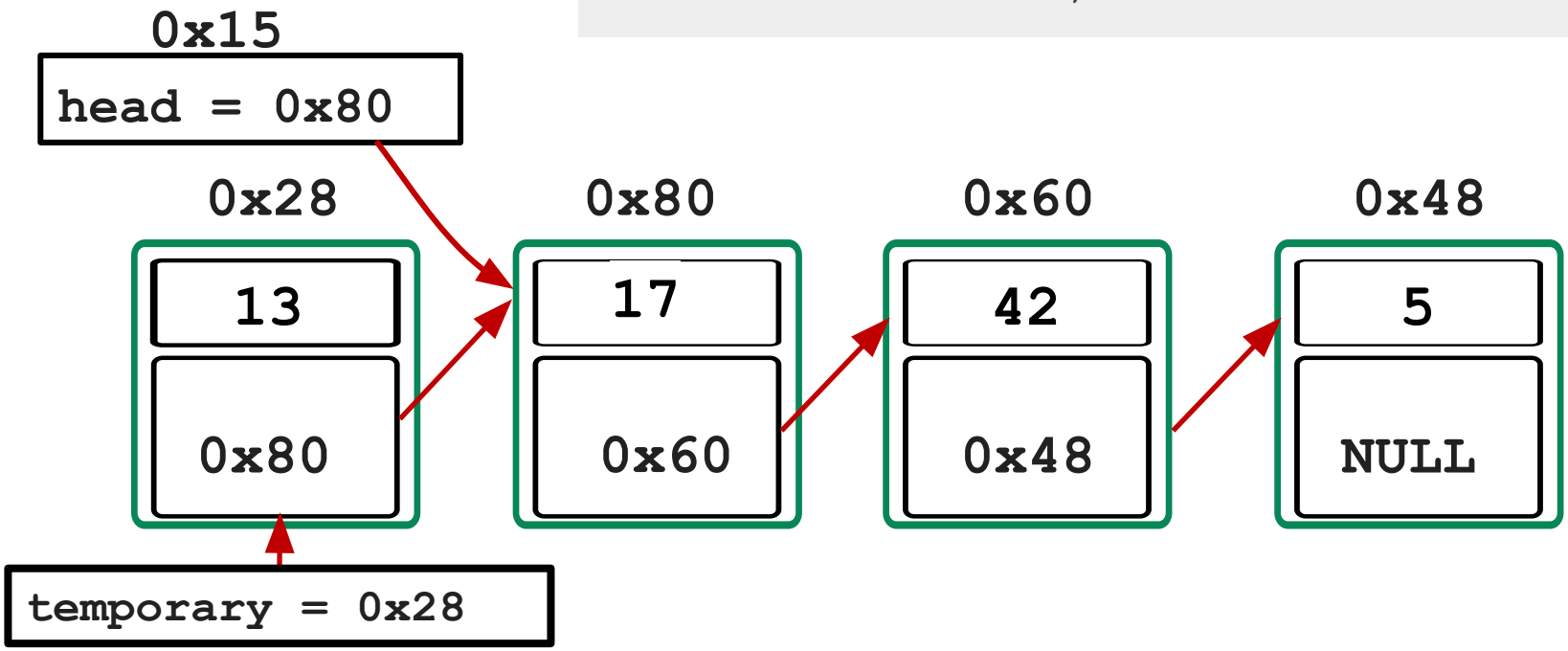
```
struct node *temporary = head;
```



# Deleting the First Node in a Linked List

Now we can update head

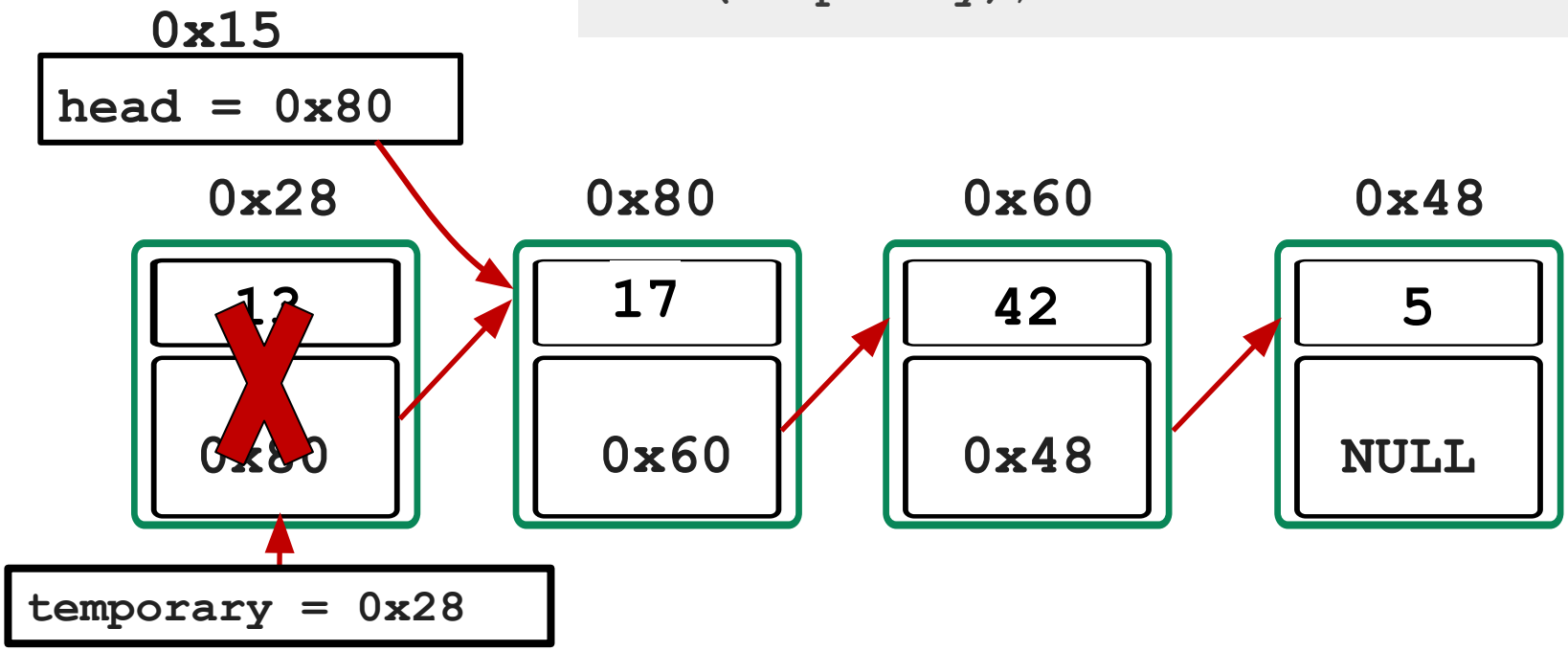
```
head = head->next;
```



# Deleting the First Node in a Linked List

Now we can free the first node

```
free(temporary);
```



# Delete All Nodes the Correct Way

Let's test it and check it with dcc --leak-check

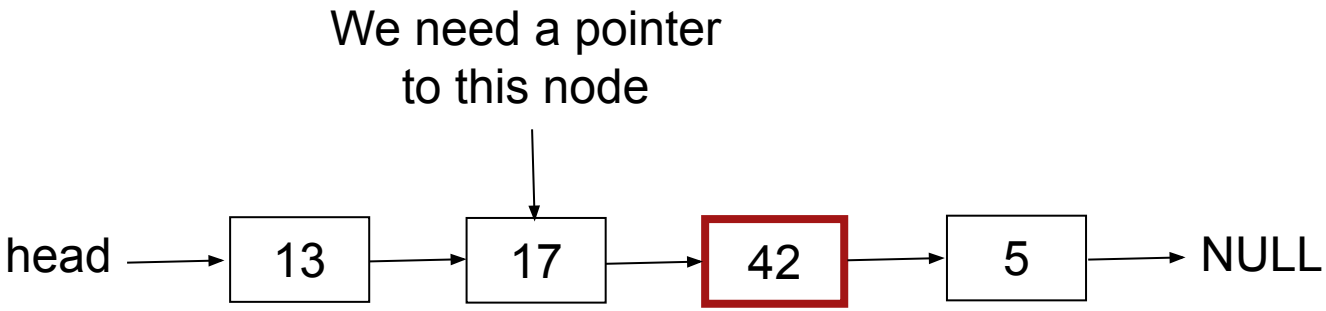
```
// Delete all nodes from a given list
void delete_all_nodes(struct node *head) {
    struct node *current = head;
    while (current != NULL) {
        head = head->next;
        free(current);
        current = head;
    }
}
```

# Search and Delete

- We want to search for a node with a particular value in it and then delete it
- Where could the item be
  - Nowhere - if it is an empty list or the list does not contain the value
  - At the head (deleting the first node in the list)
  - Between any 2 nodes in the list
  - At the tail (deleting the last node in the list)
  - There could be multiple occurrences! For now let’s just consider the first occurrence

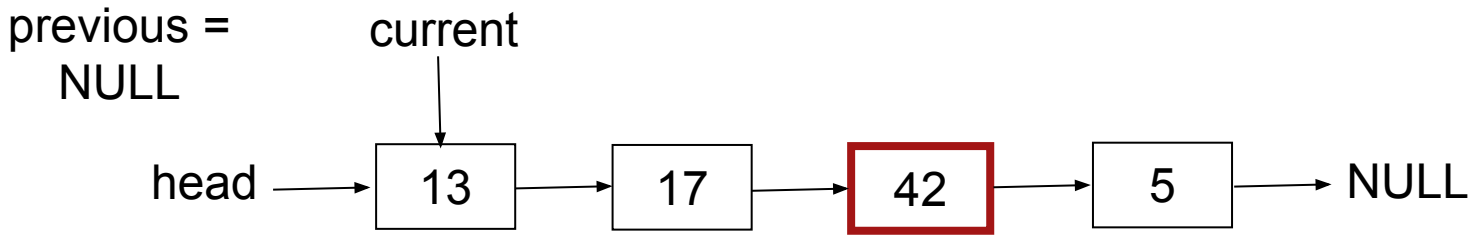
# Search and delete: between 2 nodes

- To delete a node we need to link the previous node to the next node
  - If we want to delete the node with 42, we need to find the node before it



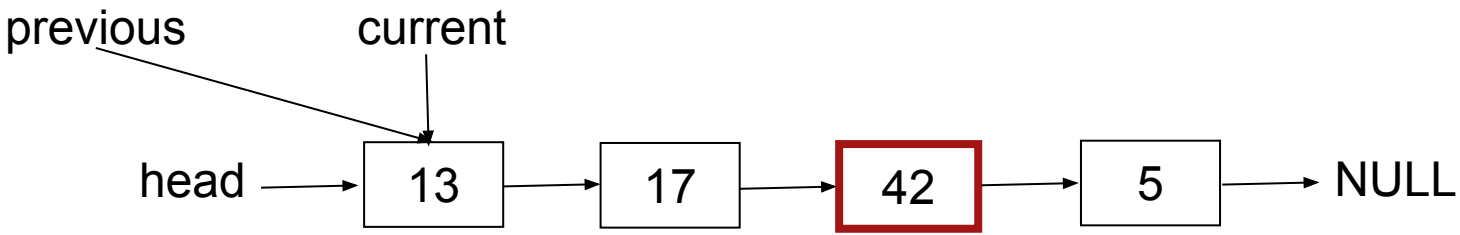
# Search and delete: between 2 nodes

```
// Approach 1: Have a previous node pointer
struct node *previous = NULL;
struct node *current = head;
while (current != NULL && current->data != search_key) {
    previous = current;
    current = current->next;
}
```



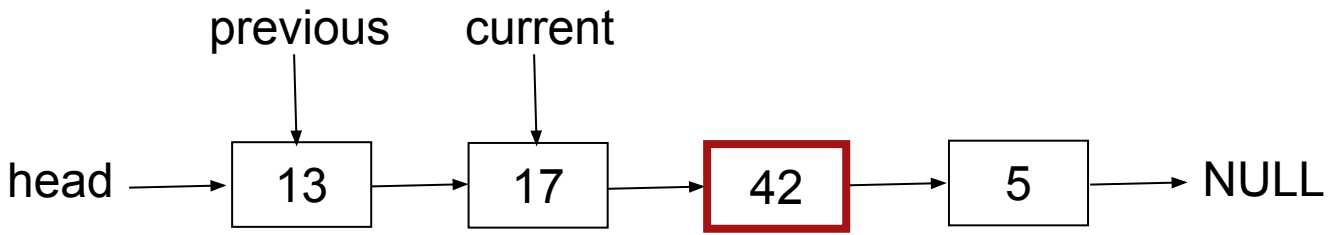
# Search and delete: between 2 nodes

```
// Approach 1: Have a previous node pointer
struct node *previous = NULL;
struct node *current = head;
while (current != NULL && current->data != search_key) {
    previous = current;
    current = current->next;
}
```



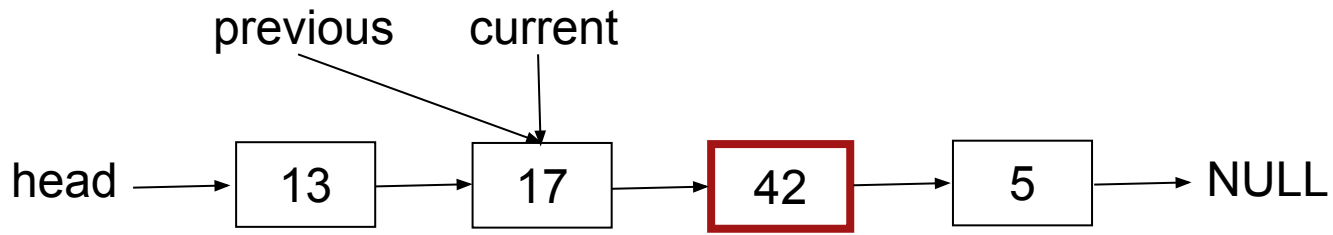
# Search and delete: between 2 nodes

```
// Approach 1: Have a previous node pointer
struct node *previous = NULL;
struct node *current = head;
while (current != NULL && current->data != search_key) {
    previous = current;
    current = current->next;
}
```



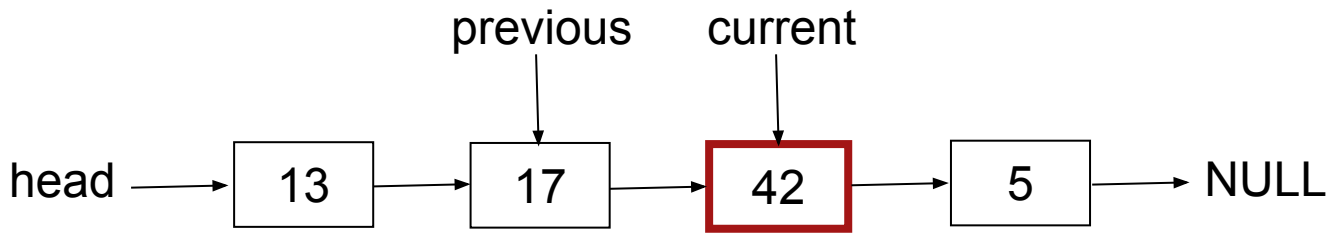
# Search and delete: between 2 nodes

```
// Approach 1: Have a previous node pointer
struct node *previous = NULL;
struct node *current = head;
while (current != NULL && current->data != search_key) {
    previous = current;
    current = current->next;
}
```



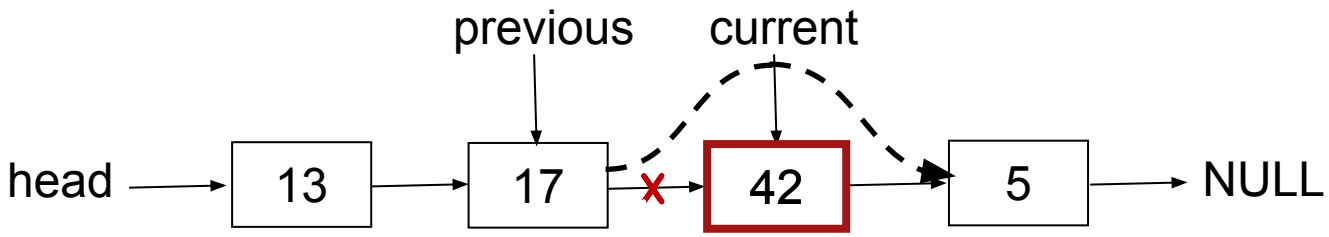
# Search and delete: between 2 nodes

```
// Approach 1: Have a previous node pointer
struct node *previous = NULL;
struct node *current = head;
while (current != NULL && current->data != search_key) {
    previous = current;
    current = current->next;
}
```



# Search and delete: Approach 1

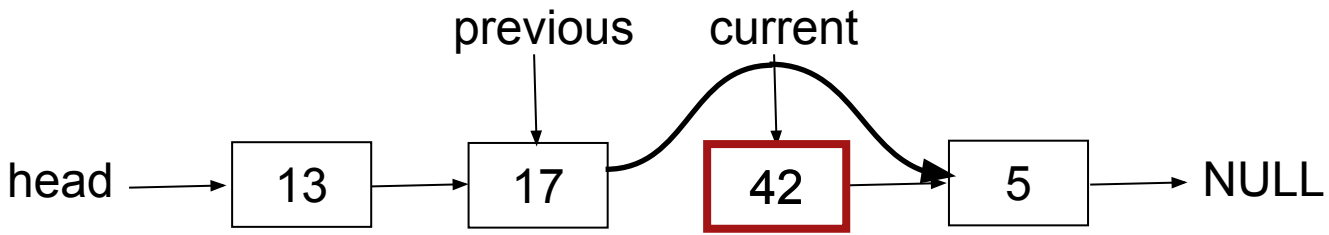
Then we need to connect current node to the one after the one we are deleting.



# Search and delete: Approach 1

Then we need to connect current node to the one after the one we are deleting.

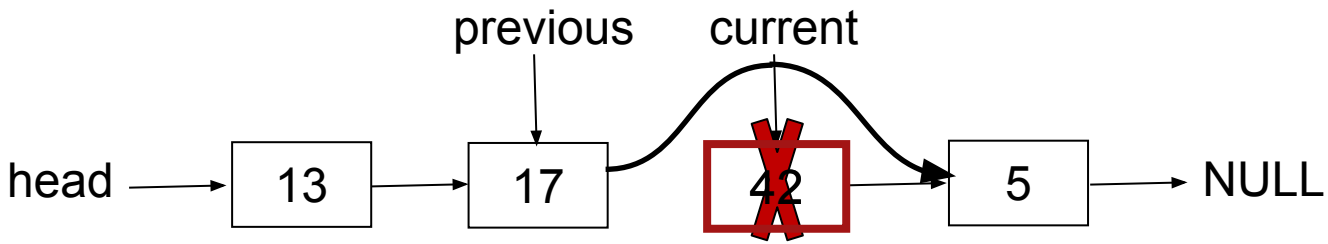
```
previous->next = current->next;
```



# Search and delete: Approach 1

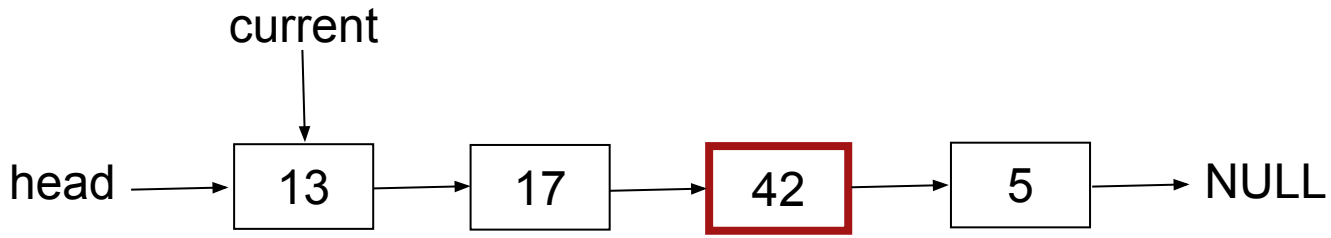
Now we can free the node we want to delete

```
free(current);
```



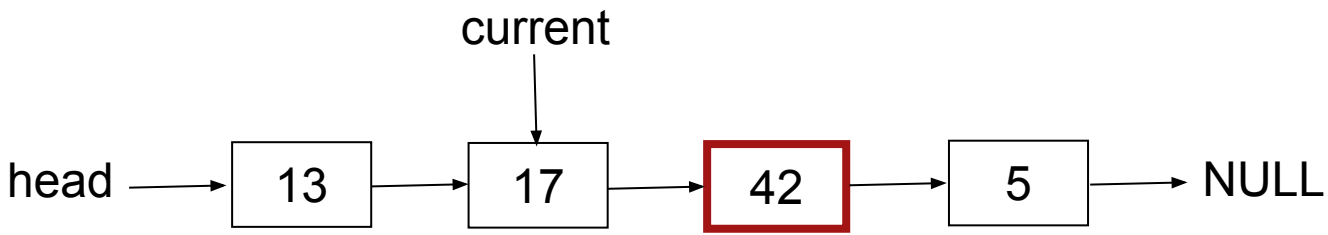
# Search and delete Approach 2: general case

```
// Approach 2: Just use 1 pointer to traverse
// but check the next node
struct node *current = head;
while (current->next != NULL &&
       current->next->data != search_key) {
    current = current->next;
}
```



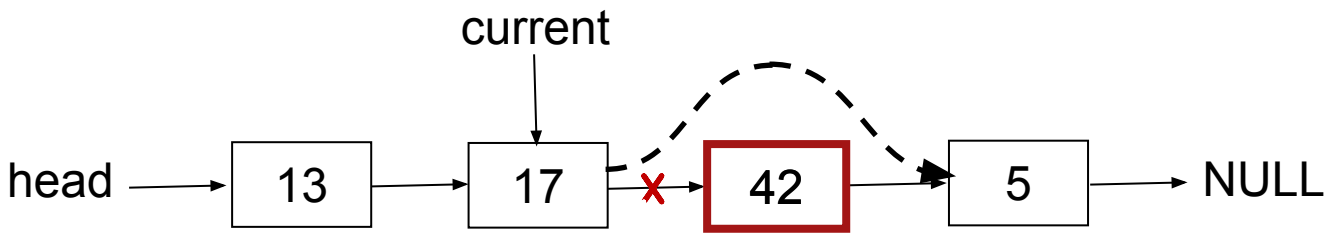
# Search and delete Approach 2: general case

```
// Approach 2: Just use 1 pointer to traverse
// but check the next node
struct node *current = head;
while (current->next != NULL &&
       current->next->data != search_key) {
    current = current->next;
}
```



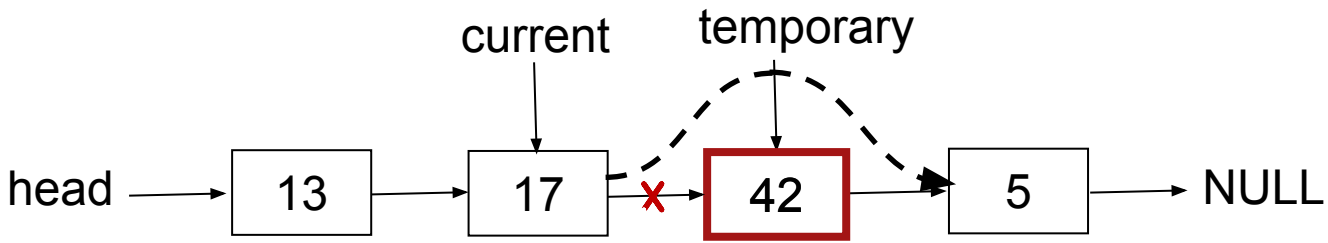
# Search and delete: Approach 2

Then we need to connect current node to the one after the one we are deleting. But we still need a pointer to the node we want to free. How can we do that?



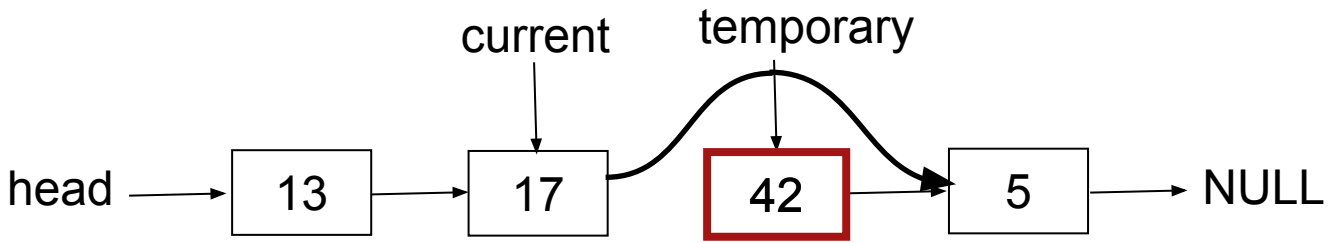
# Search and delete: Approach 2

```
struct node *temporary = current->next;
```



# Search and delete: Approach 2

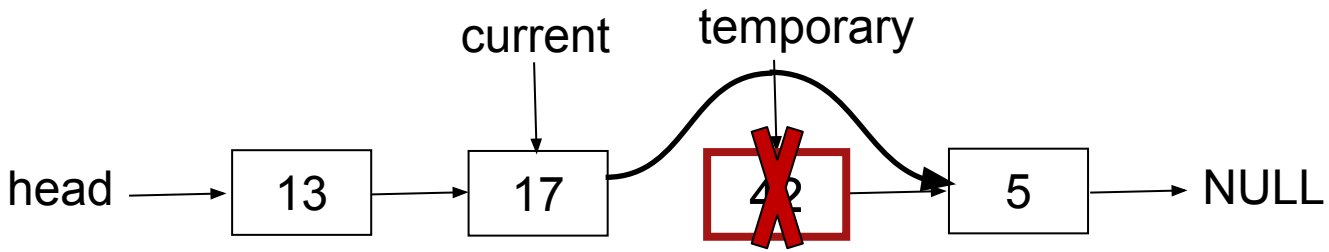
```
struct node *temporary = current->next;
current->next = temporary->next;
```



# Search and delete: Approach 2

Now we can free the node we want to delete

```
free(temporary);
```



# Coding

Let’s code up the second  approach.

Let’s extend  our first approach to delete all occurrences.

# Email Management System

# Email Management System

- Files/code provided (4 files):
  - email_management_system.c (TODO)
  - email_management_system.h (PROVIDED)
  - main.c (PROVIDED)
  - test_main.c (PROVIDED)
- Complete all `TODO` function definitions in email_management_system.c

# Before you start coding

- Understand the Problem
  - what the provided code is doing
  - how it all fits together
  - how to compile it and run the code
- Draw diagrams
  - do this before/while coding each function too!
- Think about different test cases
  - do this before/while coding each function too!

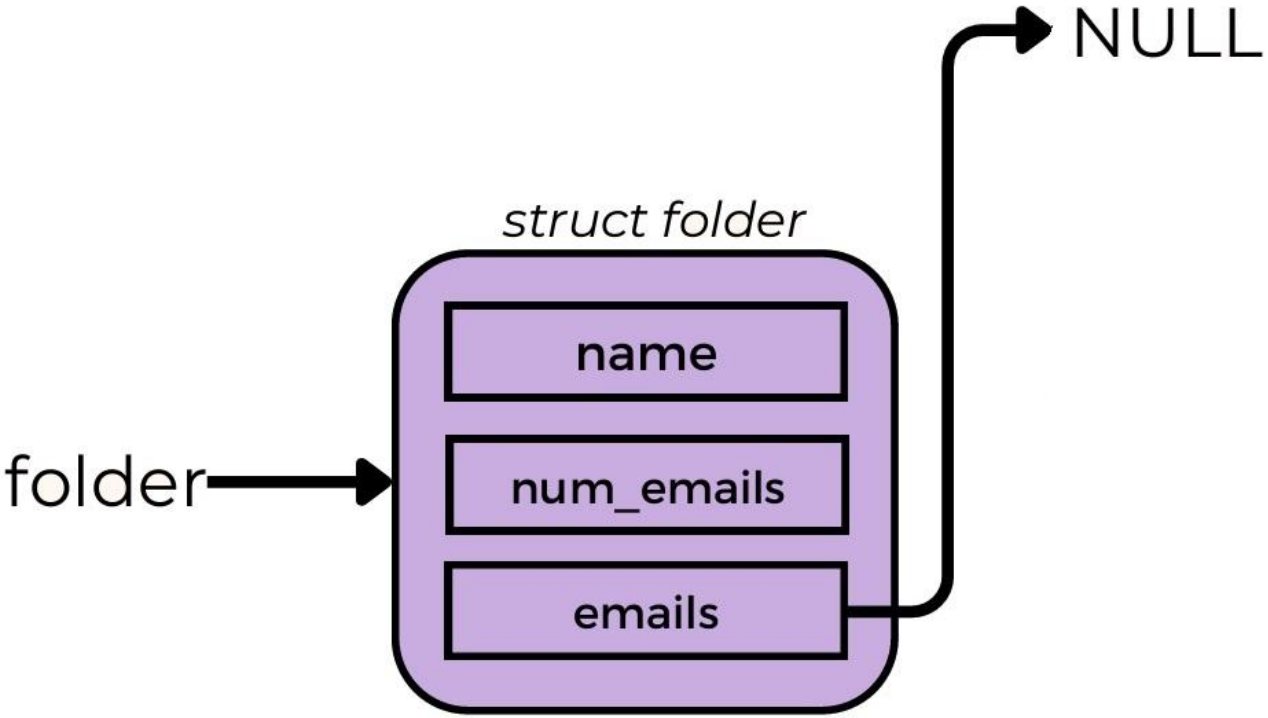
# structs

```c
struct folder {
    char name[MAX_LEN];
    //to use later :)
    //int num_emails;
    struct email *emails;
};
```

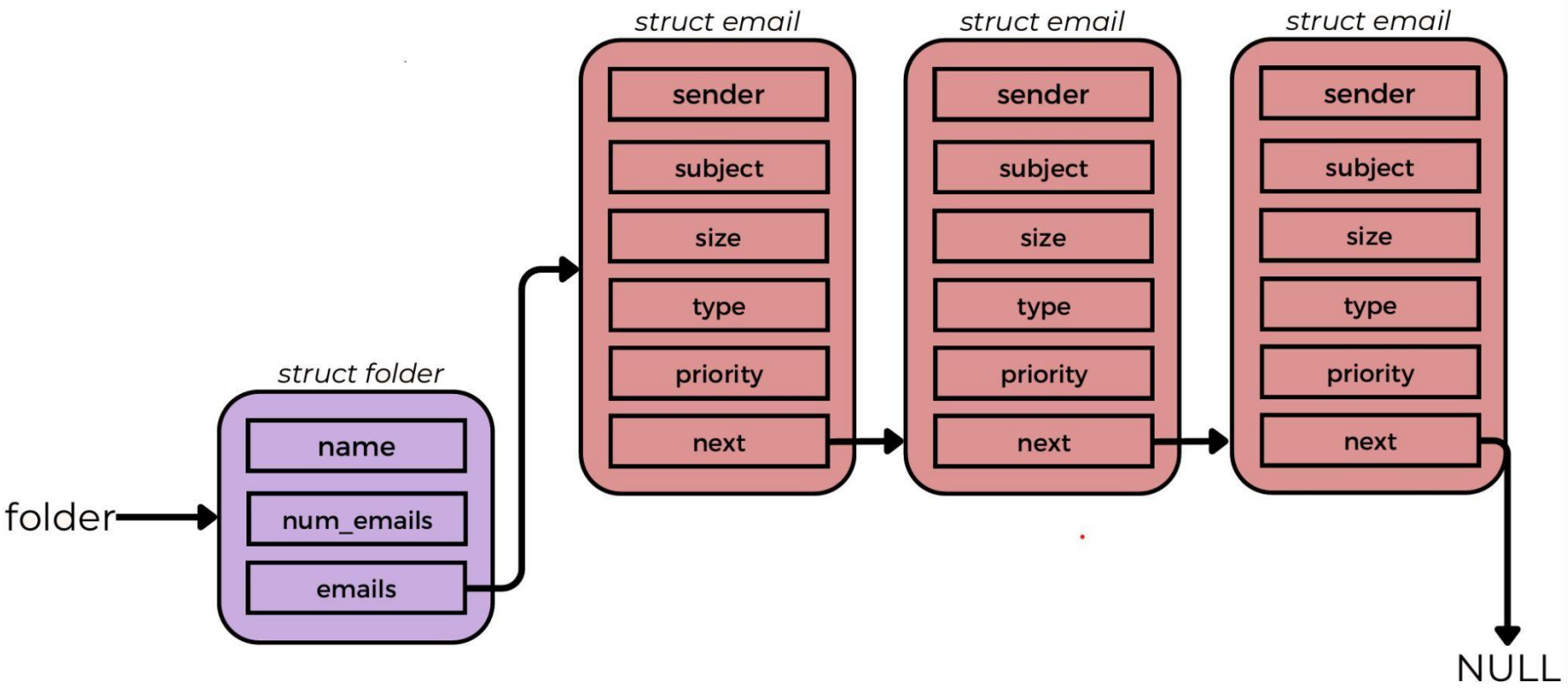
```c
struct email {
    char sender[MAX_LEN];
    char subject[MAX_LEN];
    double size;
    enum email_type type;
    enum priority_type priority;
    struct email *next;
};
```

Where are our nodes? Where is the head or list?

# Visualisation of the system

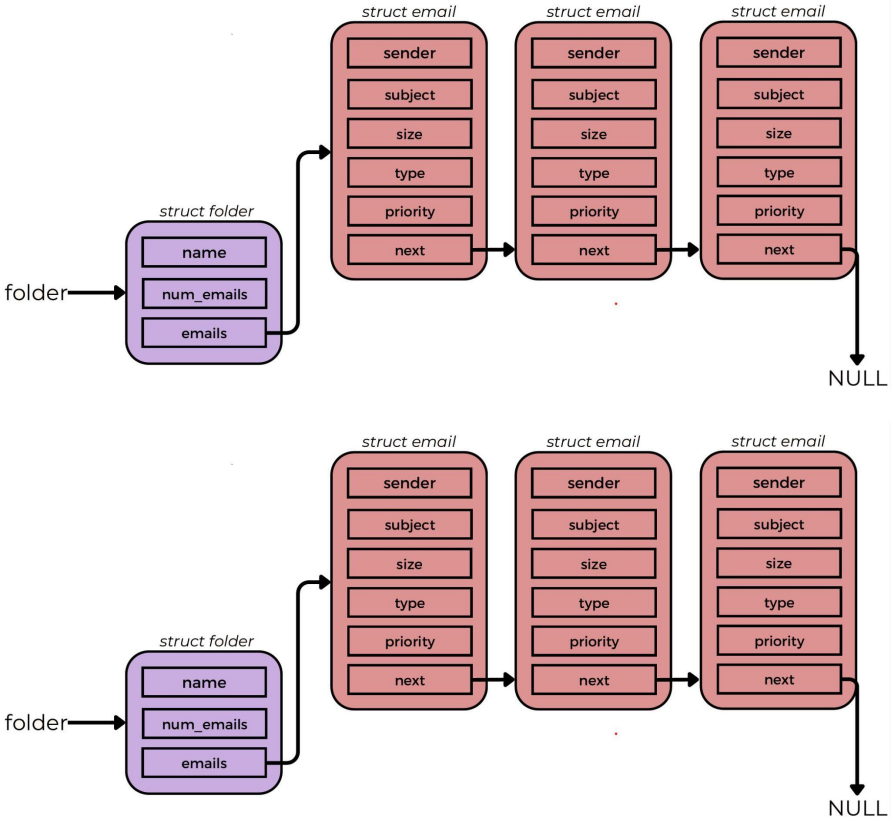


# Visualisation of the system



# Visualisation of the system

We can create many folders, each containing linked lists of emails.



# Compiling and running the code

- We have 2 files that contain main functions.
- We can only have 1 main function per program.
- We can compile and run the first program as follows:

```
dcc -o test_main test_main.c email_management_system.c
./test_main
```

- We can compile and second program as follows:

```
dcc -o main main.c email_management_system.c
./main
```

# Functions to Write

- Stage 1
  - create_folder
  - insert_email_at_head
  - search_email
- Stage 2
  - clear_folders
  - delete_email_of_priority
- Stage 3
  - merge_folders
  - split_folder

# Stage 4 Extension Stage

- Keep track of size of folder to make count_emails more efficient.
- Create an account system struct that contains a linked list of folders and an account name.
- Write code to print all folders
- Write code to print out all emails in all folders

# What did we learn today?

- Recap
  - Linked List Deletion
  - Implement search and delete approach 2
  - Extend approach 1 to delete all occurrences
- Larger Linked List Application
  - Multi-file program
  - Linked lists used inside of other structs
  - Linked lists containing complex 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/xdhUUfVSN7

# Next Lecture:

- Continue with Linked List Application
- Exam Information
- Revision

# Reach Out

Content Related Questions:

**Forum**

Admin related Questions email:

**cs1511@unsw.edu.au**