COMP1511/1911 Programming Fundamentals

Week 8 Lecture 2

Linked Lists
Deletion and Application

Week 8 Check off: Mini Exam

- Full details in forum post
- **In person attendance**
  - If you are in an online class OR if you cannot make your timetabled in-person class
    - sign up for an in-person lab time and attend in the second hour
    - attend your regular tlb as well if you are an online student
  - Email course account if for some reason you are sick or can't attend on the day cs1511@unsw.edu.au

Week 8 Check off: Mini Exam

- **Format: 1 hour**
  - A one dot array question
  - A two dot array question
  - Two debugging questions
- Attempt and submit at least one questions to get **check off**
- There are also regular lab exercises to do for the lab

This is a great opportunity to try out the exam environment and attempt questions that will be the same style as the array hurdles and debugging questions in the final exam!

Revision Sessions This Week

Wednesday 22/07/2026 4PM-6PM in **Kora Lab J17-307.**

Sign up and get more details on forum.

Topic: pointers, strings, linked lists

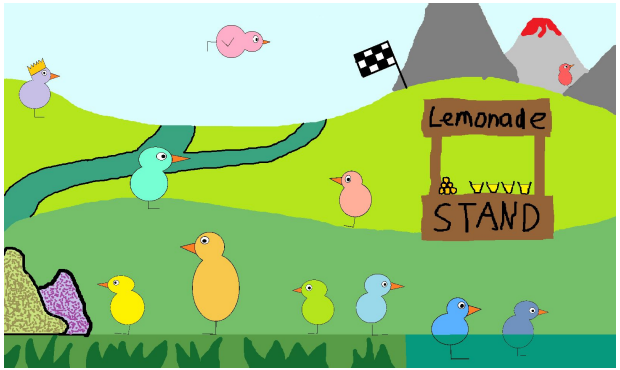
Assignment 2: CS Duck Life 🦆

Assignment Due Date: 7th August 6:00 PM (Friday Week 10)

We have an Assignment 2 Video 📽️ to help you get started.

We have Help Sessions and labs to help you too!

If you would like to organise a **drop in** session to extra 30 min 1-1 help with a tutor please email **cs1511@unsw.edu.au**



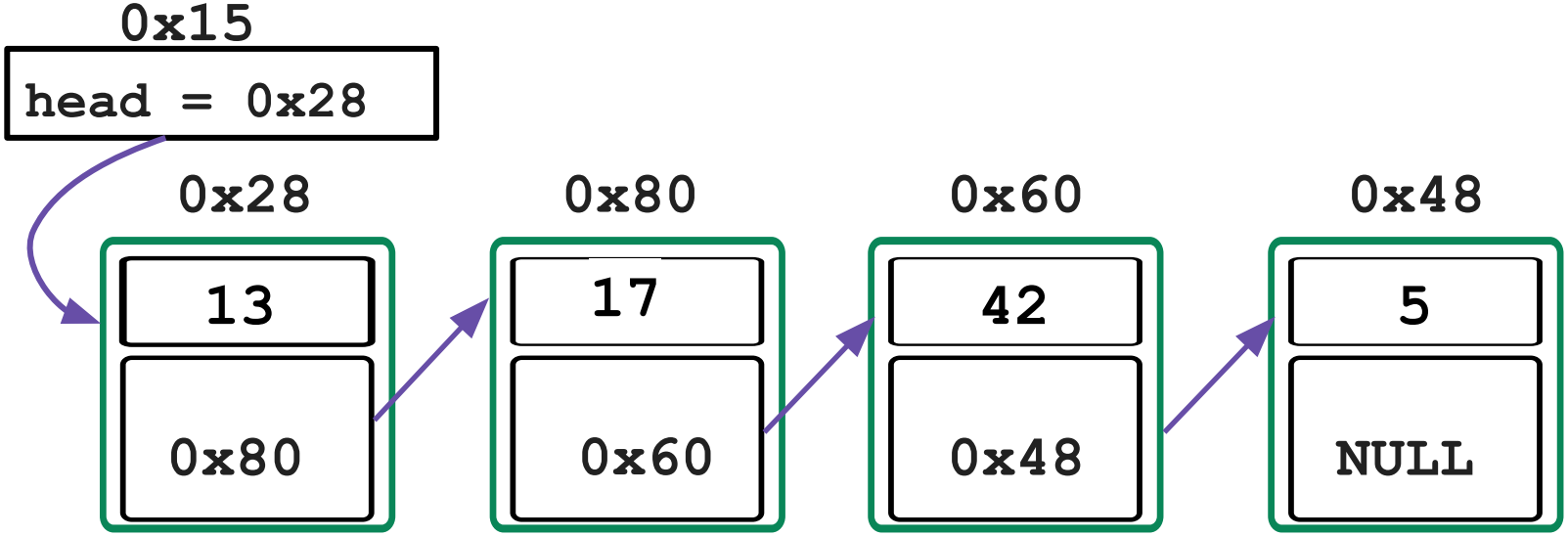
Today's Lecture

- https://cgi.cse.unsw.edu.au/~cs1511/26T2/code/week_8/
- Recap:
  - Deletion and free
- Linked list deletion
  - Delete all occurrences
- Linked Lists a Larger Application
  - Linked Lists as fields in other structs
  - Linked Lists with more complex data (other than just int)
  - Helpful for assignment 2

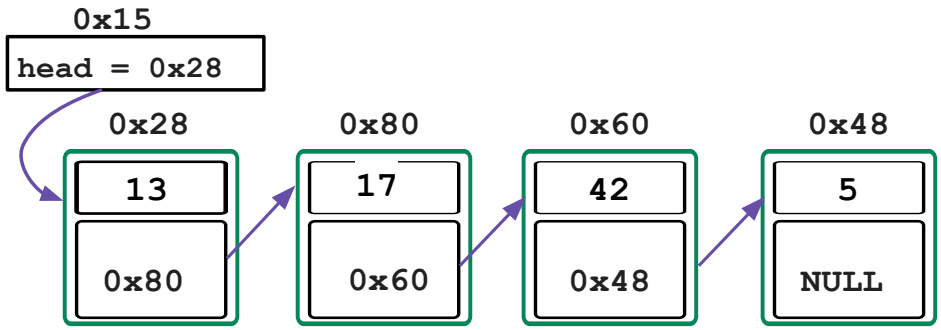
Linked List Recap

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.



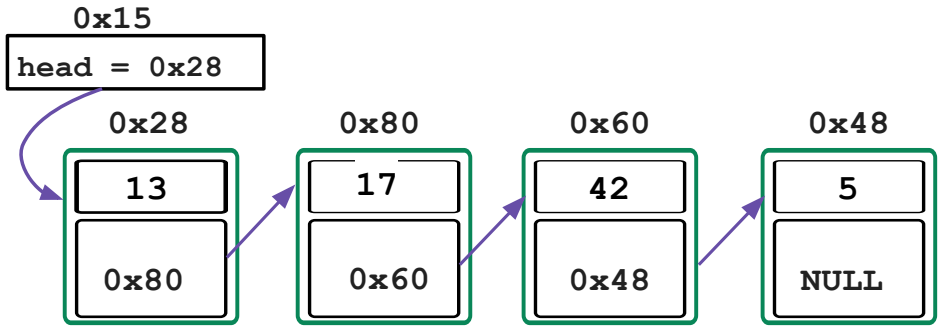
What are the problems with these solutions?




```c
//Solution 1
free(head);
head = head->next;
return head;
```

```c
//Solution 2
head = head->next;
free(head);
return head;
```

Which is correct out of these solutions?



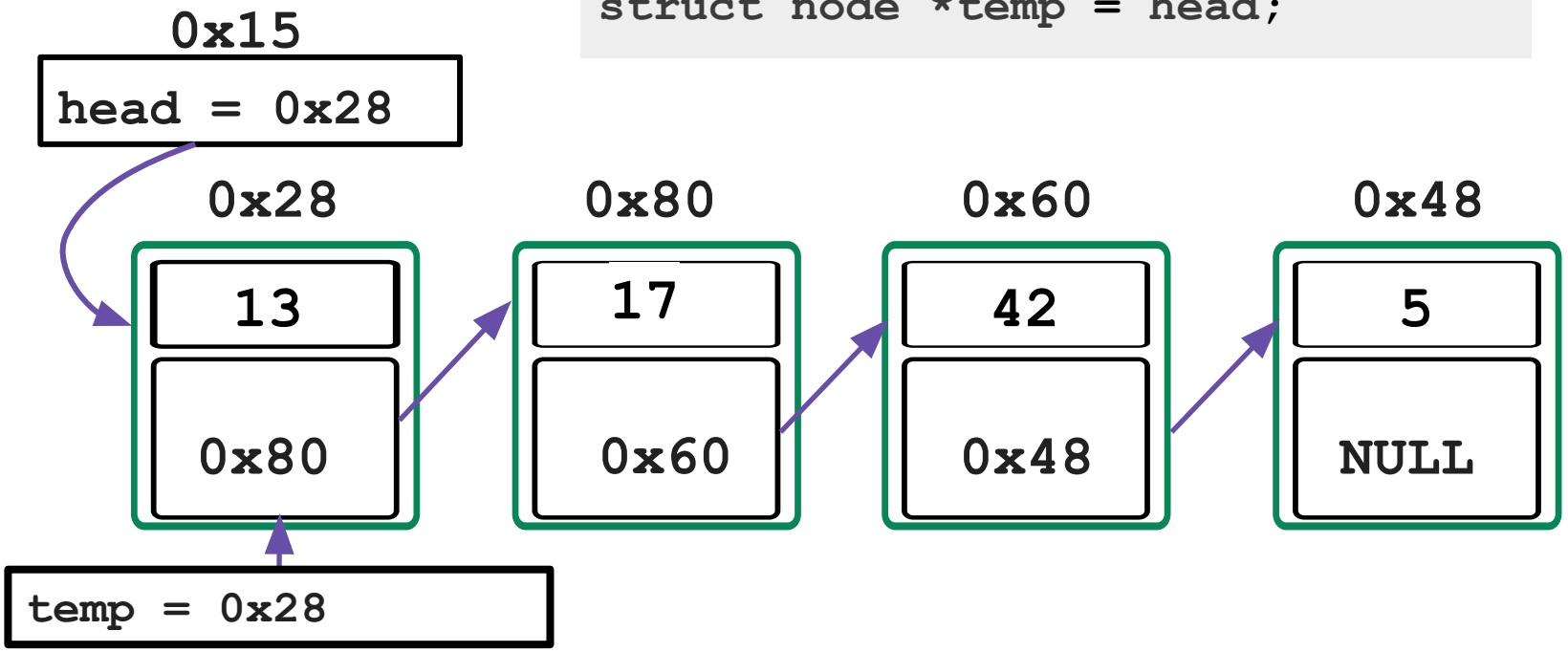

```c
//Solution 3
struct node *temp = head;
head = head->next;
free(temp);
return head;
```

```c
//Solution 4
struct node *temp = head;
free(temp);
head = head->next;
return head;
```

Deleting the First Node in a Linked List

Let's create a pointer to the first node

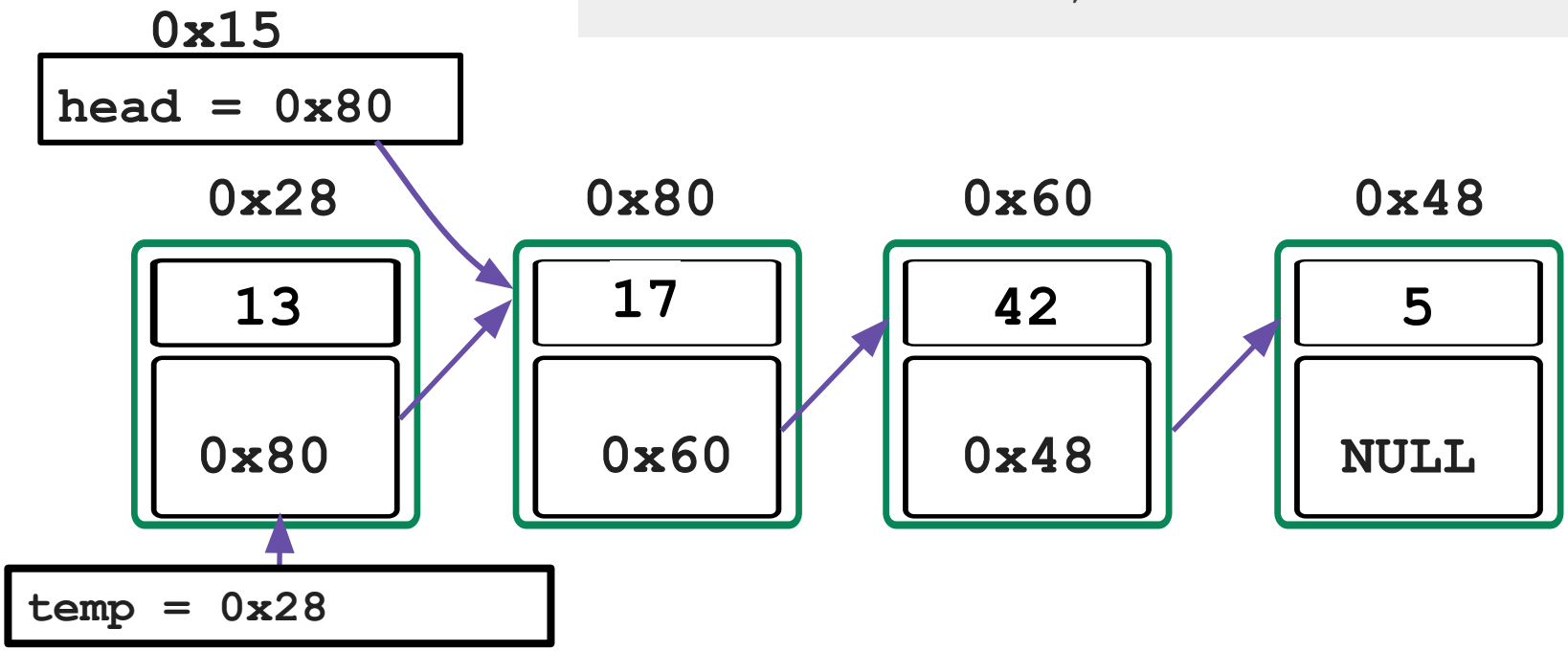
```
struct node *temp = head;
```



Deleting the First Node in a Linked List

Now we can update head

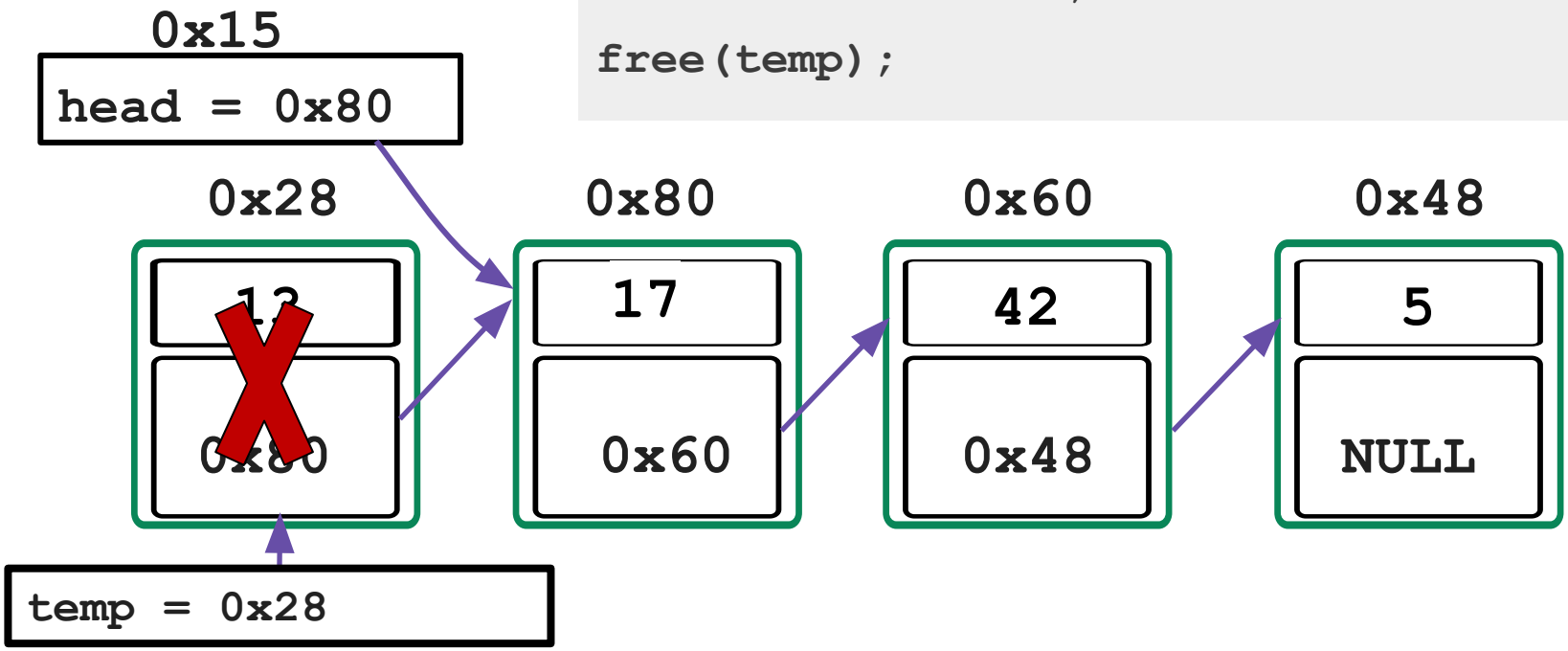
```
head = head->next;
```



Deleting the First Node in a Linked List

Now we can free the first node

```
head = head->next;
free(temp);
```



Deleting the First Node in a Linked List

We need a special case to make for empty lists, otherwise the line that does head = **head->next** will cause the program to crash since head would be NULL.

```
struct node *delete_first_node(struct node *head) {
    if (head == NULL) {
        return NULL;
    }
    struct node *temp = head;
    head = head->next;
    free(temp);
    return head;
}
```

Free List the Correct Way

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

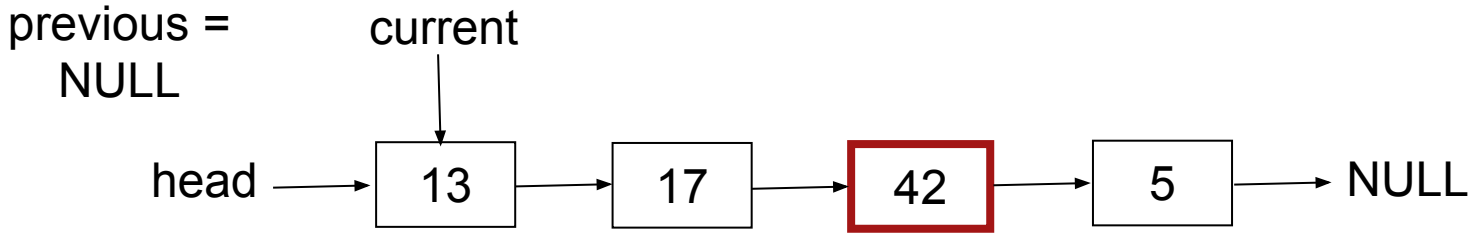
```
// Delete all nodes from a given list
void free_list(struct node *head) {
    struct node *current = head;
    while (current != NULL) {
        head = head->next;
        free(current);
        current = head;
    }
}
```

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

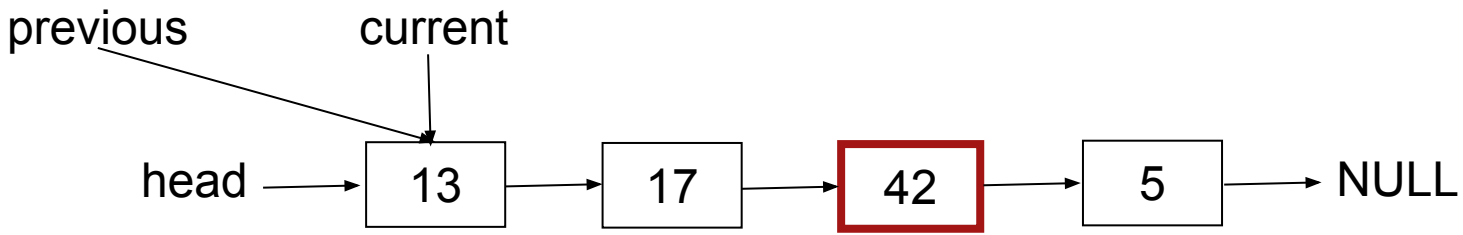
Search and delete: Approach 1

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



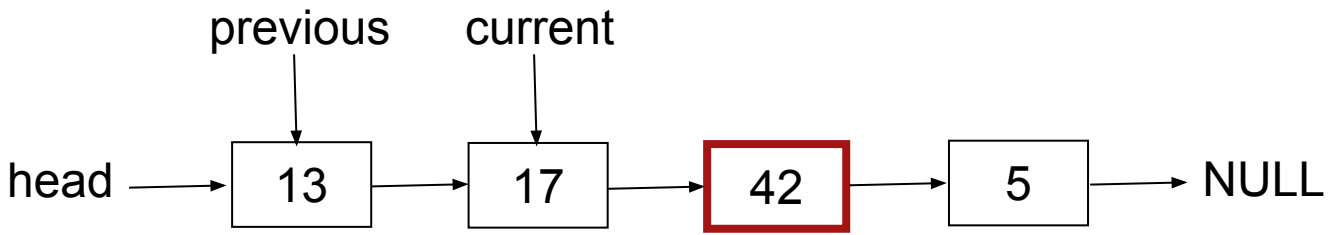
Search and delete: Approach 1

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



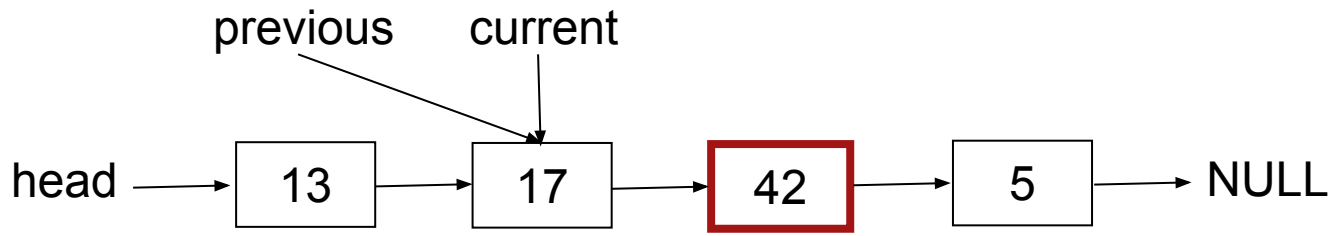
Search and delete: Approach 1

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



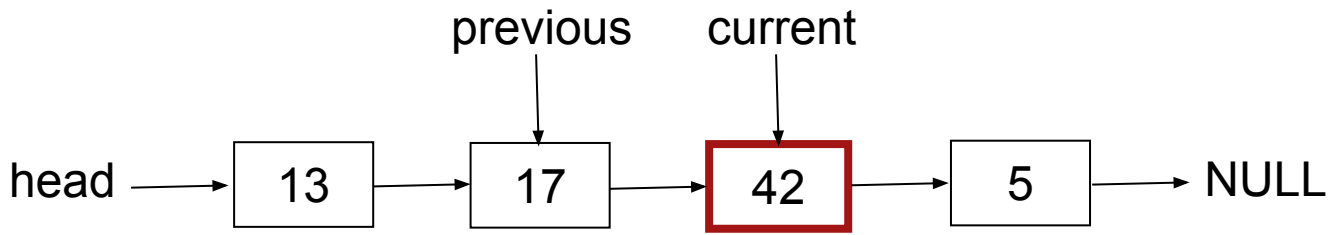
Search and delete: Approach 1

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



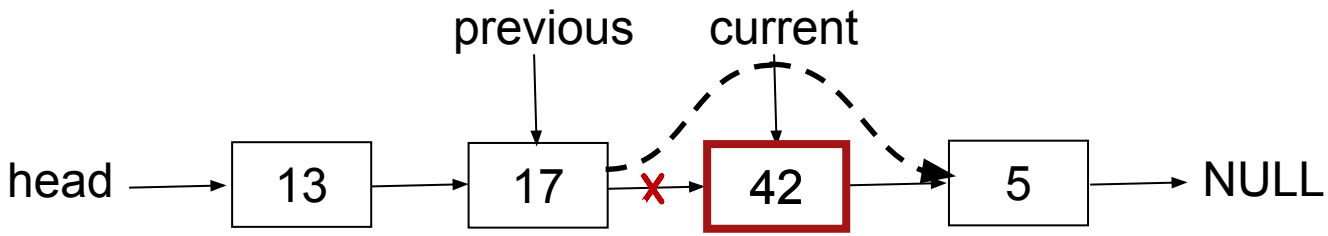
Search and delete: Approach 1

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



Search and delete: Approach 1

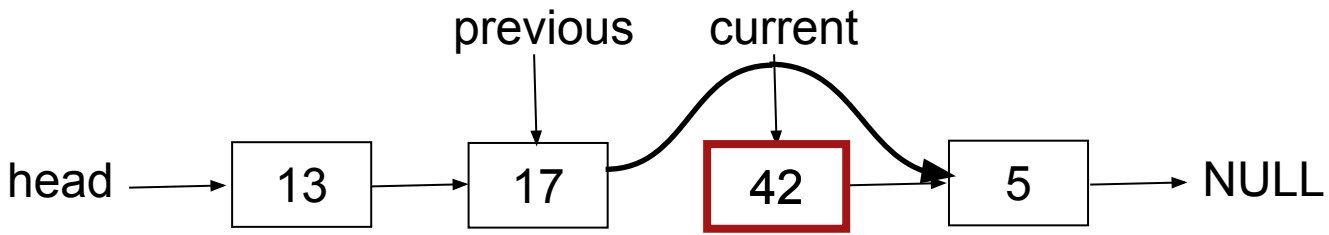
Then we need to connect previous node to the one after the one we are deleting.



Search and delete: Approach 1

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

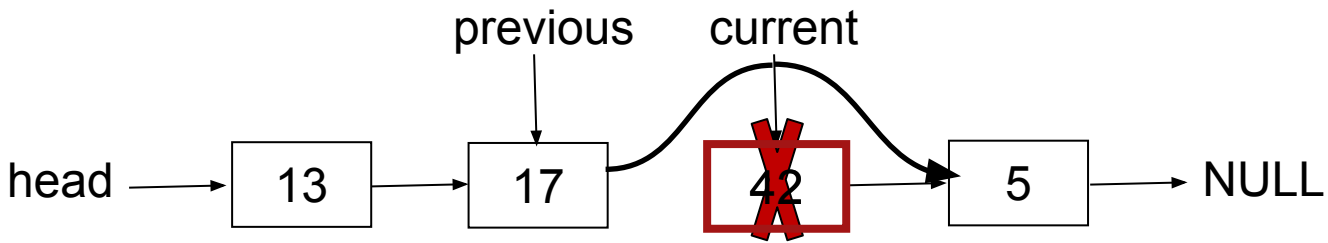
```
previous->next = current->next;
```



Search and delete: Approach 1

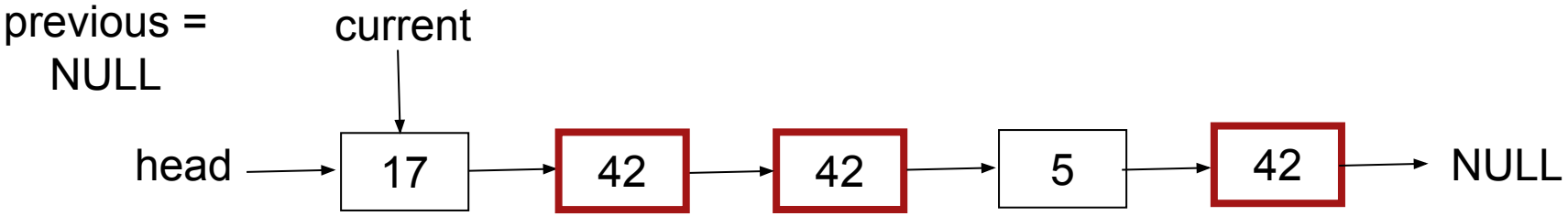
Now we can free the node we want to delete

```
free(current);
```



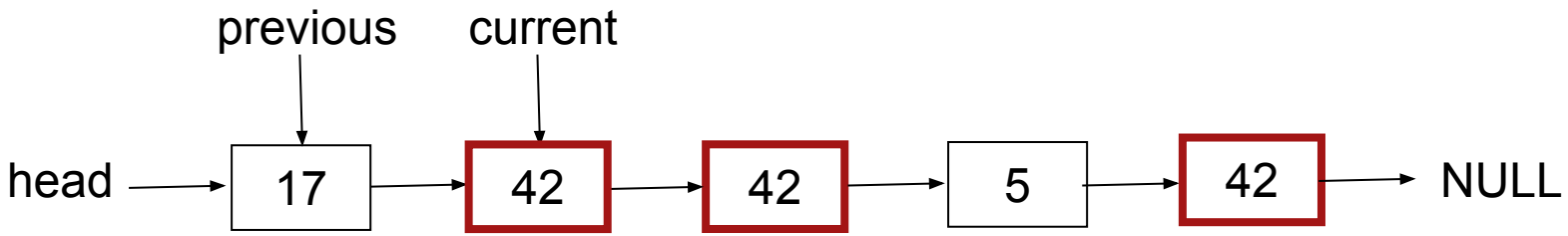
Exercise: Search and Delete All

Modify our search and delete to delete all occurrences.



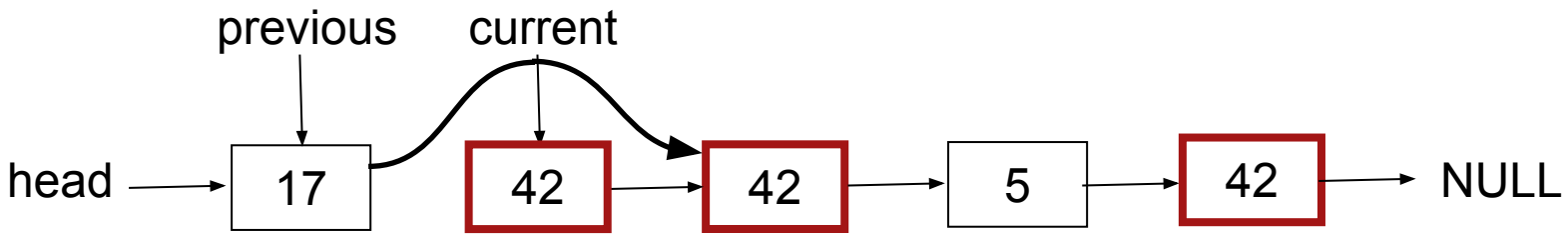
Exercise: Search and Delete All

Modify our search and delete to delete all occurrences.



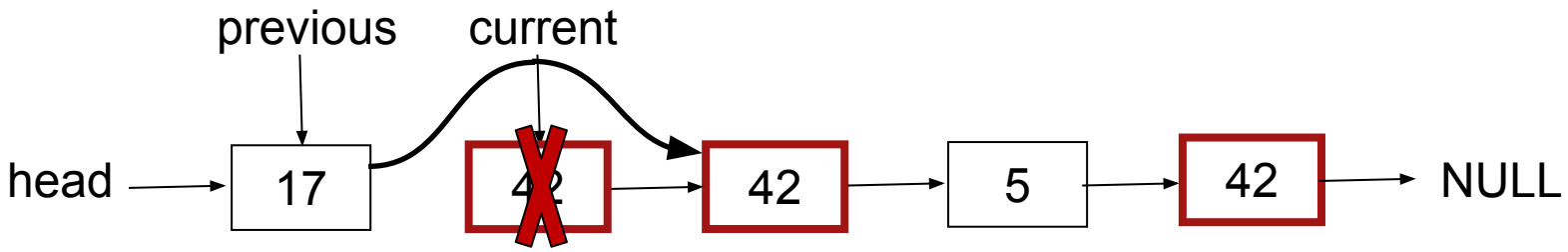
Exercise: Search and Delete All

Modify our search and delete to delete all occurrences.



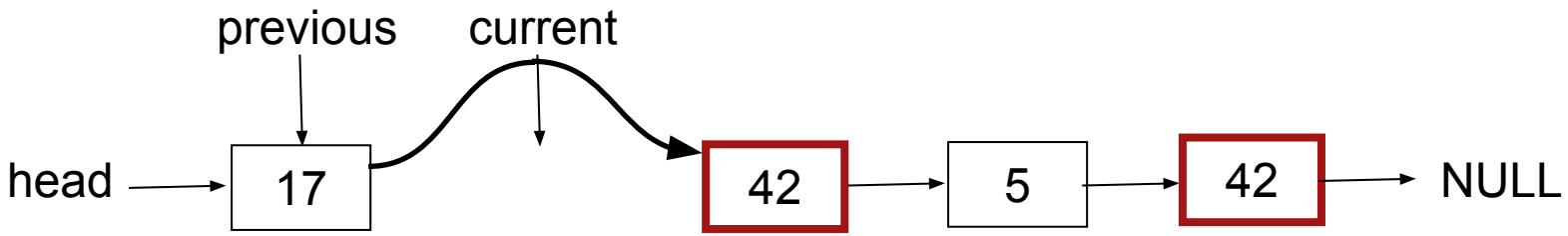
Exercise: Search and Delete All

What do we do with previous and current once we delete the first node?



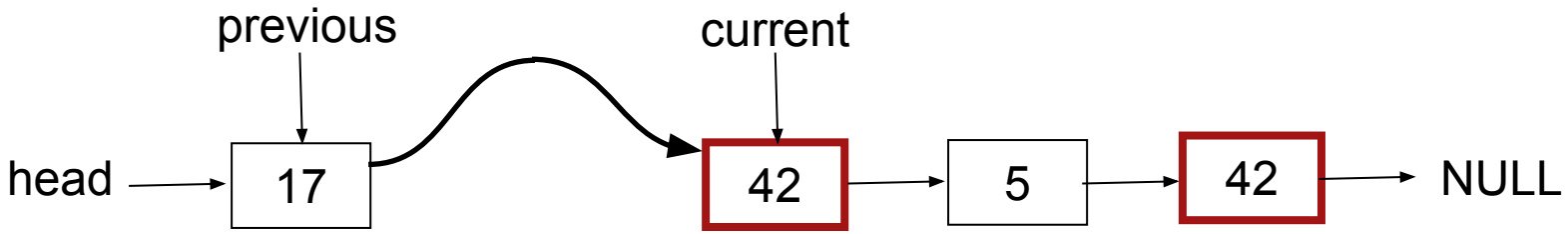
Exercise: Search and Delete All

What do we do with previous and current once we delete the first node?



Exercise: Search and Delete All

What do we do with previous and current once we delete the first node?




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

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!

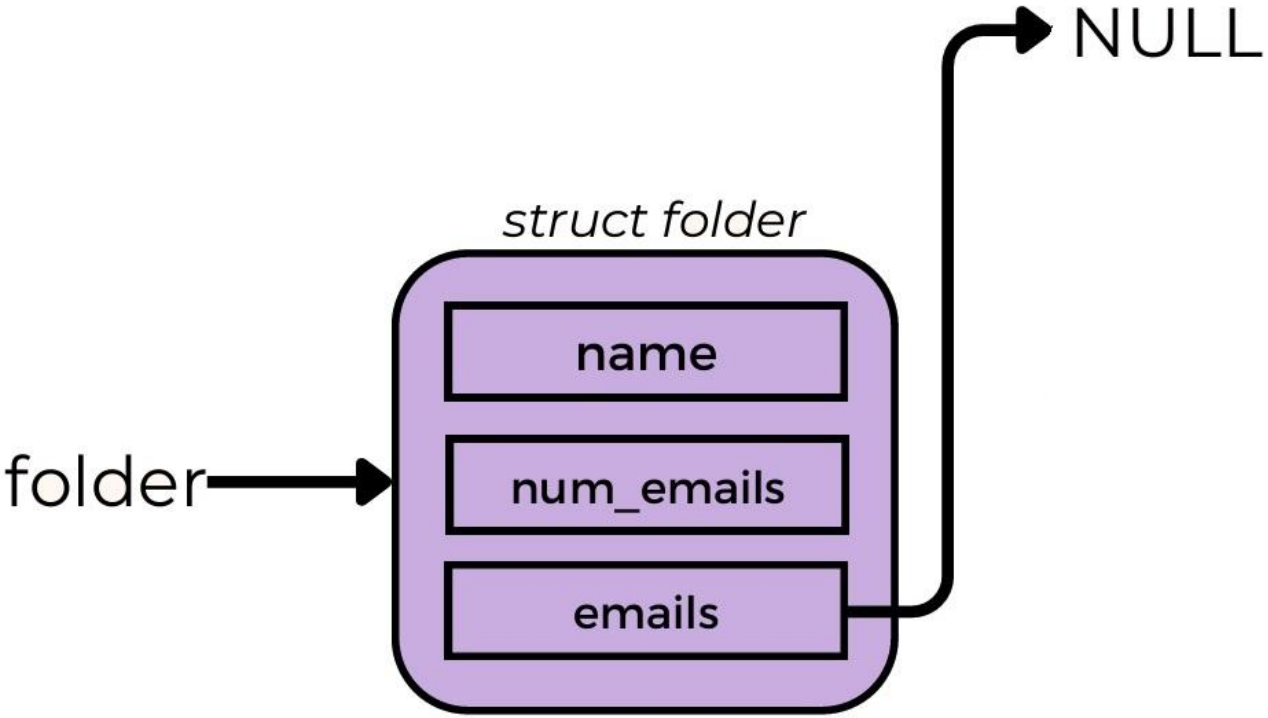
structs

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

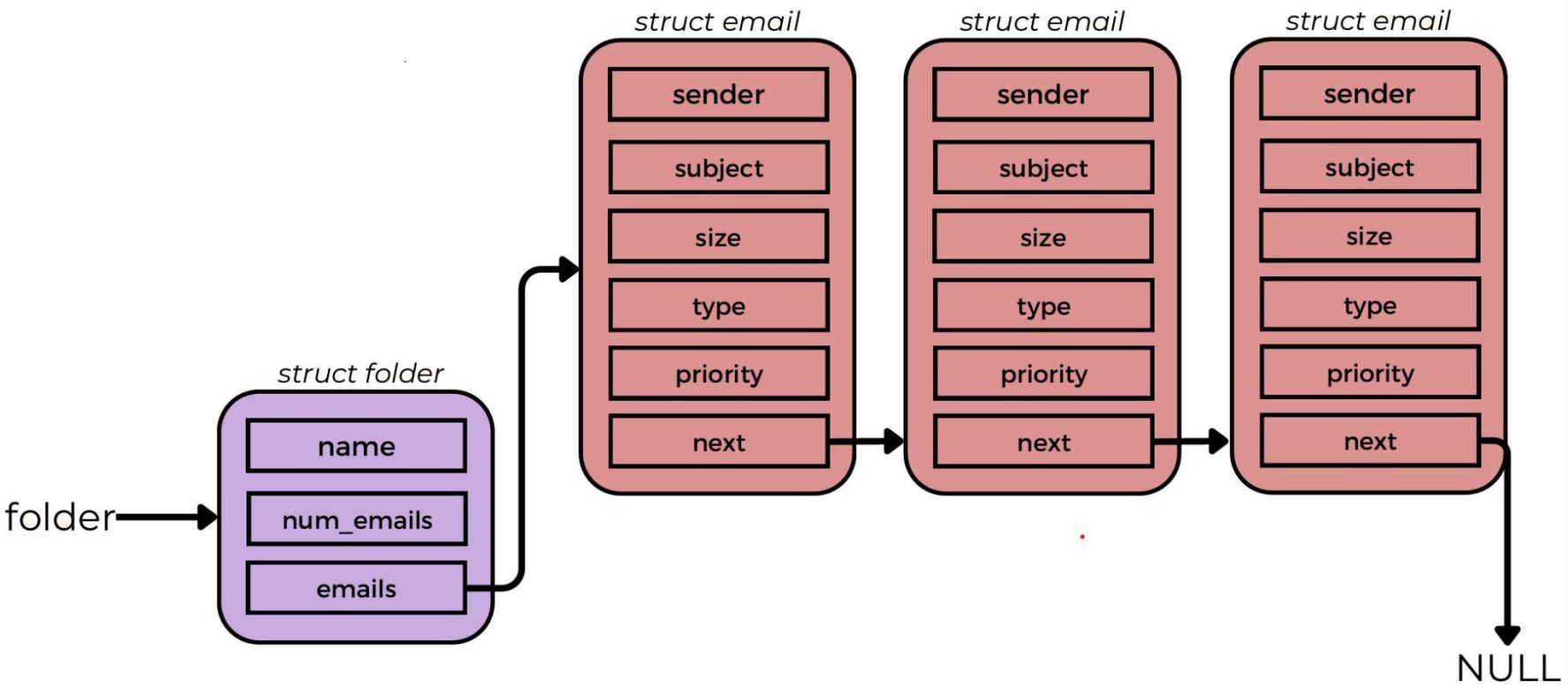
```
struct email {
    char sender[MAX_LEN];
    char subject[MAX_LEN];
    double size;
    enum email_type type;
    enum priority_type priority;
    struct email *next;
};
```

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

Visualisation of the system

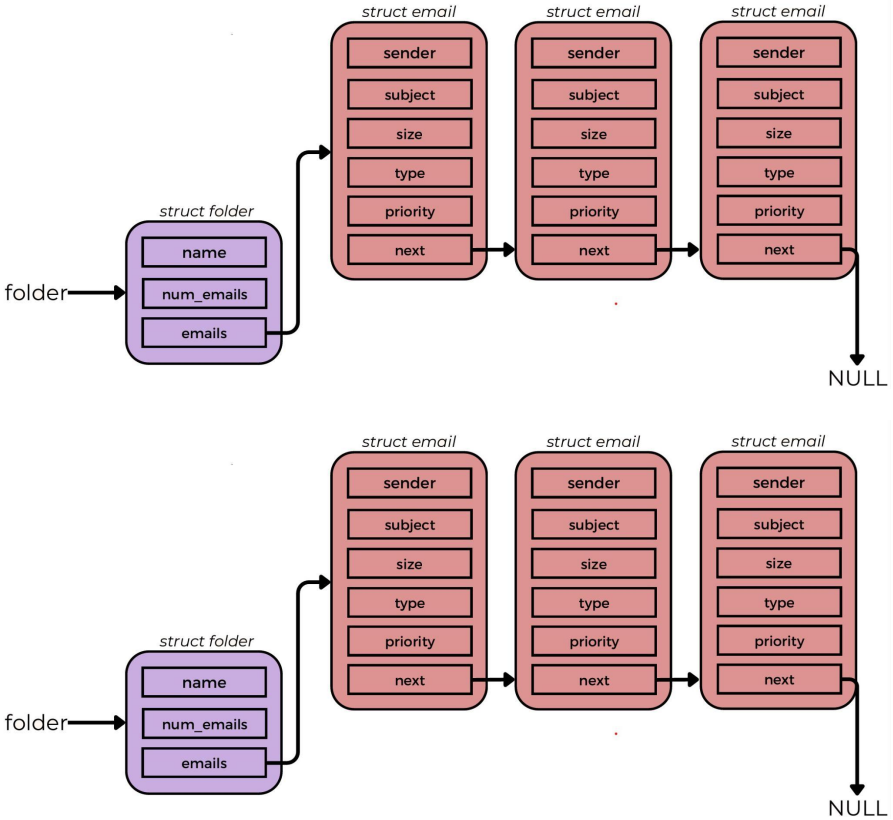


Visualisation of the system



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 Deleting elements
  - Delete First Node
  - Search and delete
- Linked Lists a Larger Application.
  - Linked Lists as fields in other structs
  - Linked Lists with more complex data (other than just int)
  - Helpful for assignment 2

Next Lecture Henry is Back!!!!

Enjoy the rest of the course and your knowledge of linked lists!