COMP1511/1911 Programming Fundamentals

Week 8
Lecture 2

Week 8 Check-Off

- Lab exams this week
  - Online students please register for an in-person one
    - Link to register: https://buytickets.at/comp1511unsw/2401097
    - Please arrive in the second hour of the lab time
  - This is a great chance to get the feel for an exam environment
  - 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

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

Week 8 Check-Off

- 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 Session

- Thursday Week 8 10-12pm in Bongo Lab G07A K17 (CSE)
  - Tutorial-style revision of course content so far, great chance to catch up!
- Sign up here
  - https://buytickets.at/comp1511unsw/2396767

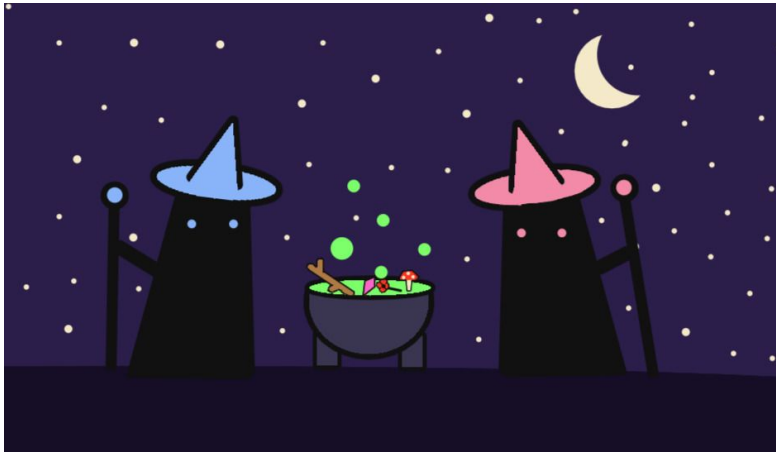


Drop In Sessions

- Chance to get 1-on-1 guidance with anything course related
- Contact cs1511@unsw.edu.au if you want to find out more or book a session

Assignment 2: CS Alchemy

- Assignment: Due Friday Week 10 at 6:00 pm (20/11/2026)
- Watch the Assignment Walkthrough Video to get started!
- Ask questions on the forum and in labs and Help Sessions

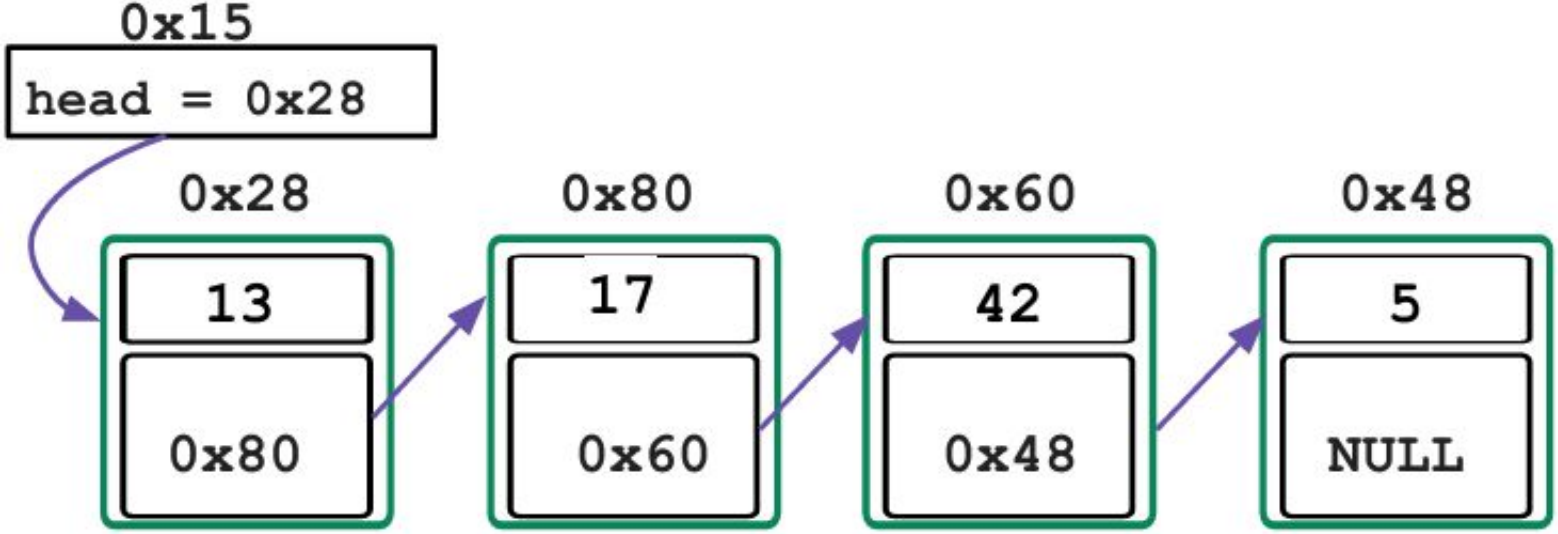


Today

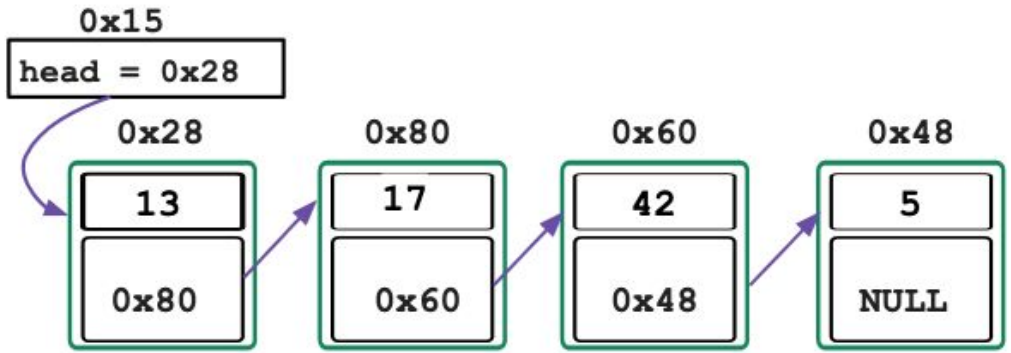
- Recap
  - Deletion and freeing
- Linked list deletion
  - Delete all occurrences
- Linked list larger application
  - Linked lists as fields in others structs
  - Linked lists with more complex data
  - Practicing for Assignment 2

Deleting the First Node

- 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



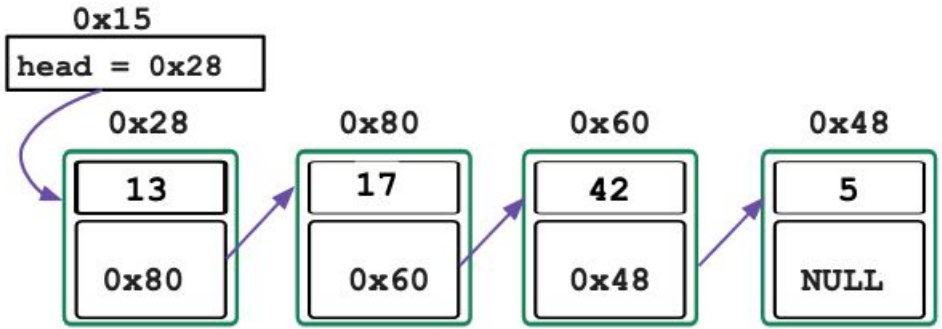
Do These Work?




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

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

Do These Work?

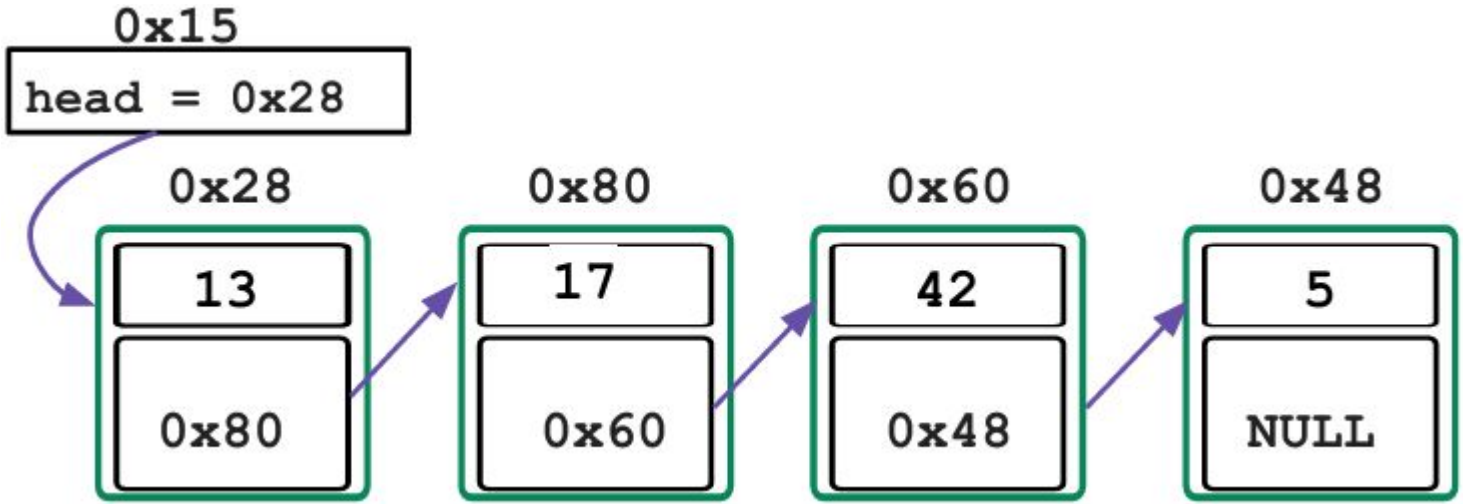



```
//Solution 3
struct node *temp = head;
head = head->next;
free(temp);
return head;
```

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

Deleting the First Node in a Linked List

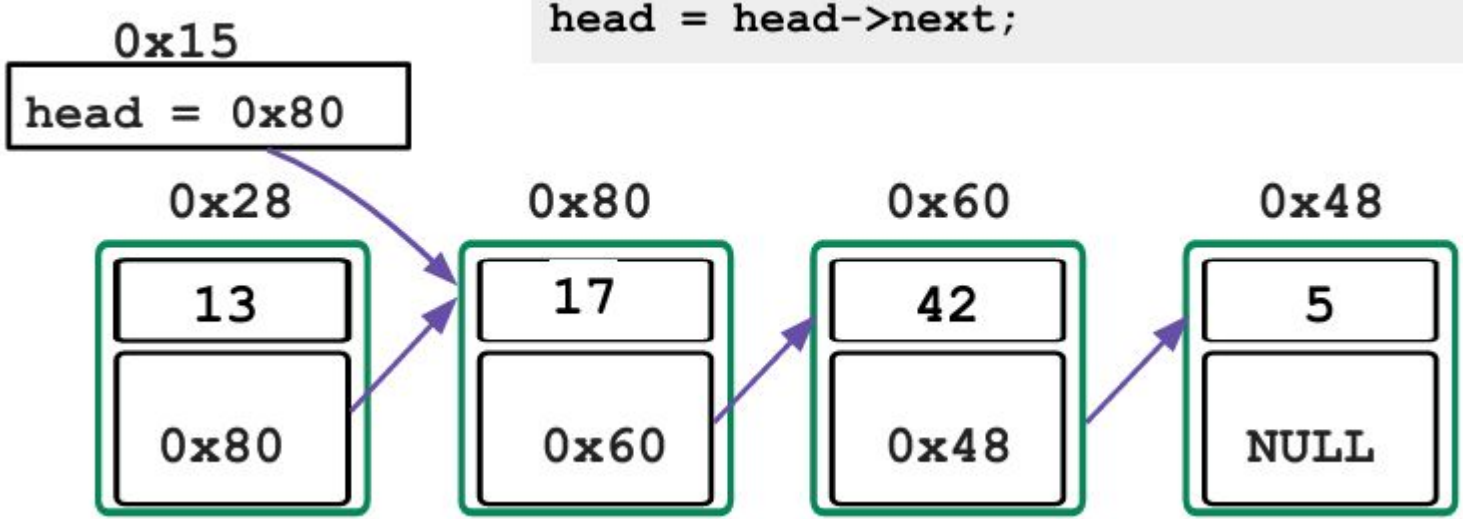
- If our list is not empty, we want to make the second node the new head of the list, and free the first node



Deleting the First Node in a Linked List

- What would be the problem with updating head first?

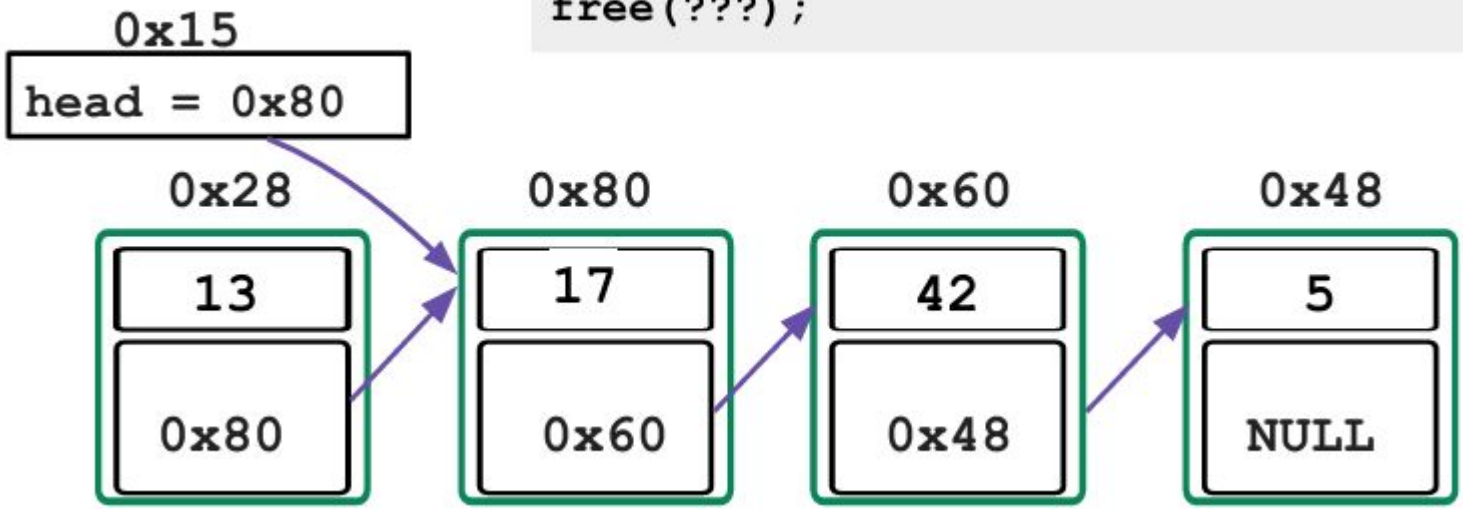
```
head = head->next;
```



Deleting the First Node in a Linked List

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

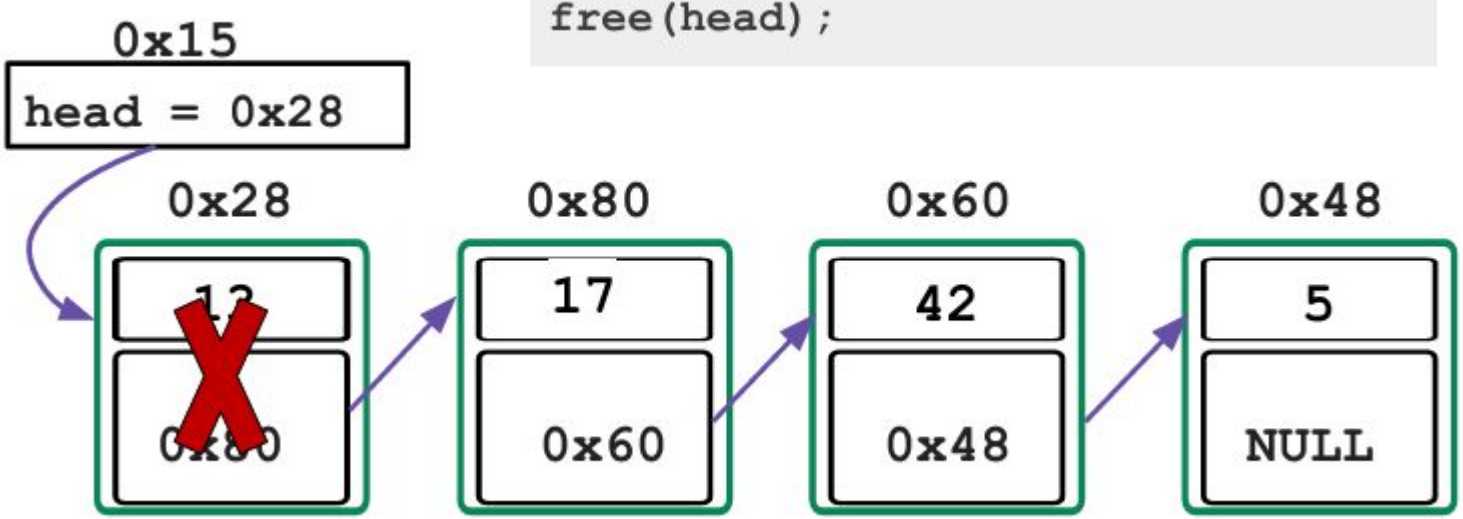
```
free(???);
```



Deleting the First Node in a Linked List

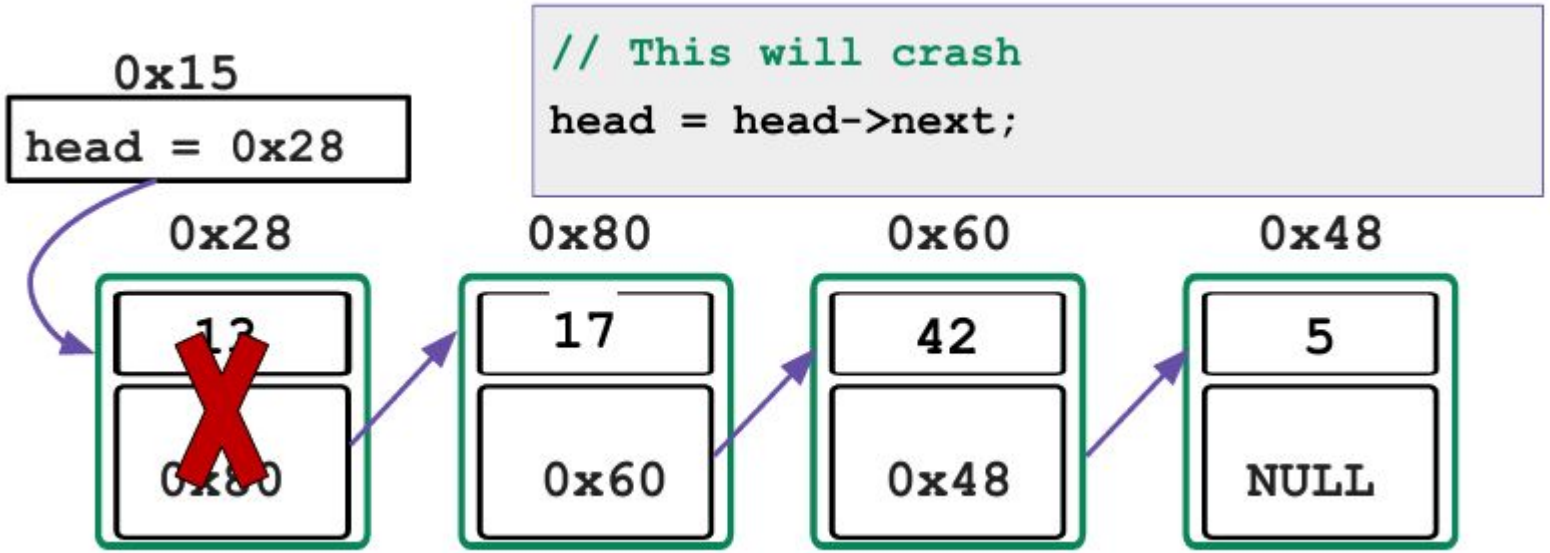
- What would be the problem calling free on head first?

```
free(head);
```



Deleting the First Node in a Linked List

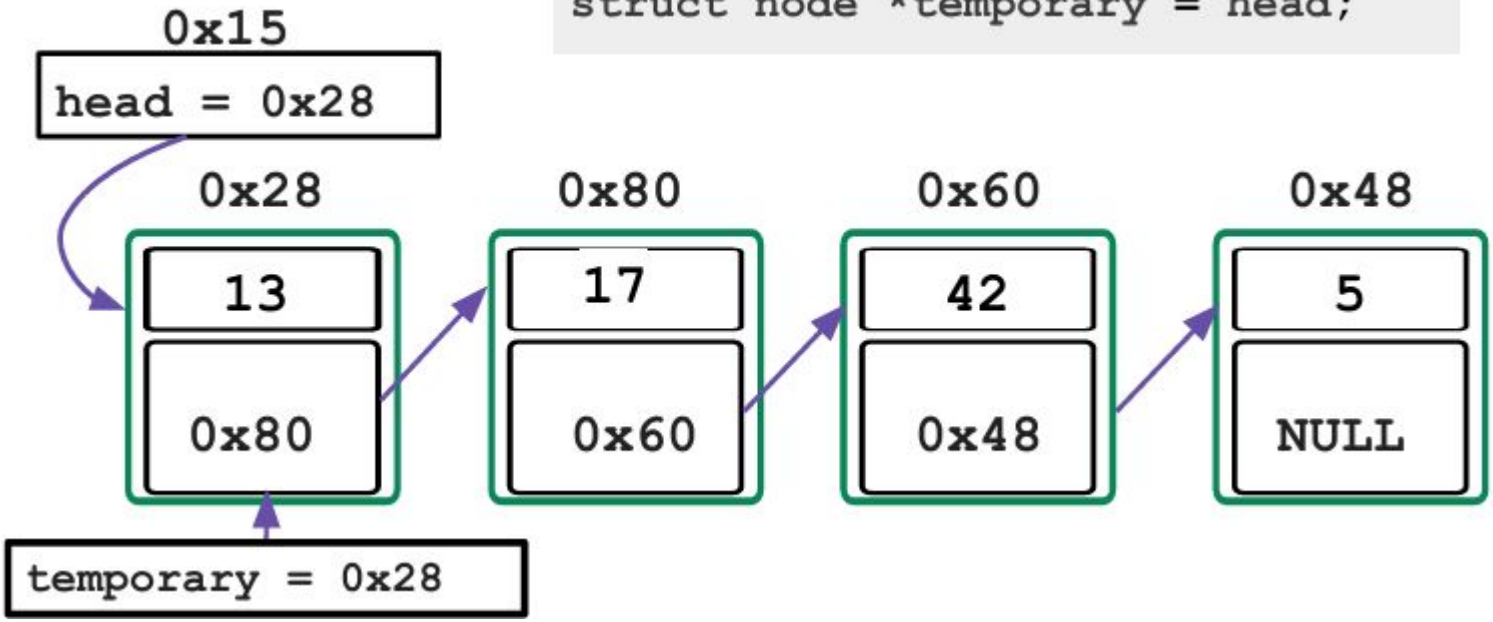
- We can’t access memory that has been freed!
  - We have lost the rest of the list



Deleting the First Node in a Linked List

- Let’s create a pointer to the first node

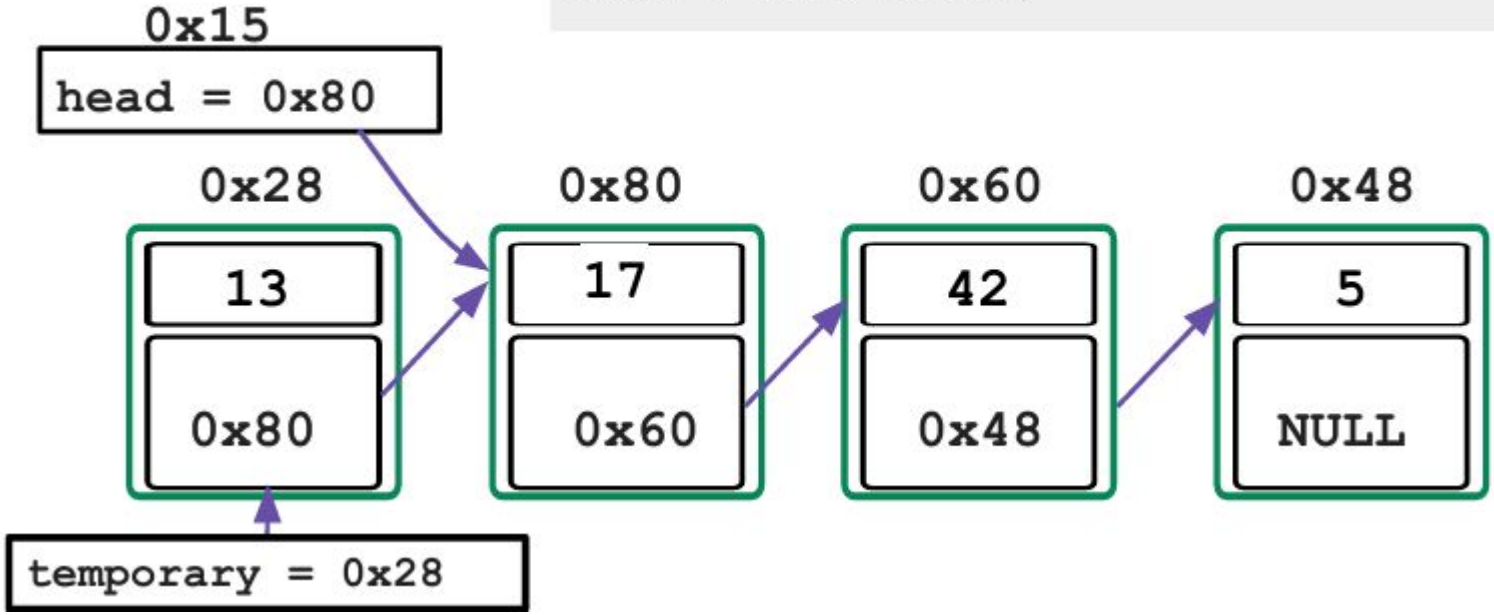
```
struct node *temporary = head;
```



Deleting the First Node in a Linked List

- Now we can update head

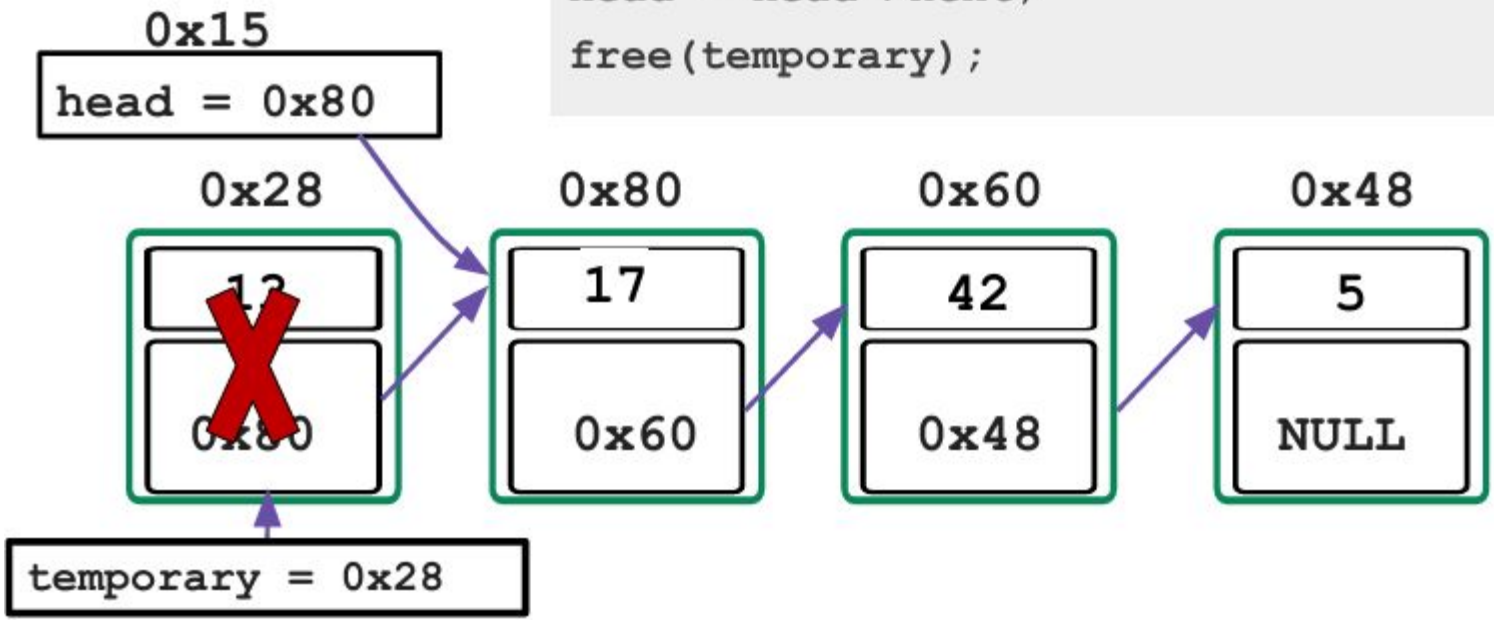
```
head = head->next;
```



Deleting the First Node in a Linked List

- Now we can free the first node

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



Deleting the First Node in a Linked List

- We need to consider the case where the list is empty
  - If it's empty
    - We can't delete anything
  - We just return the head of the list, which is NULL

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

Free Lists the Wrong Way

- What is wrong with this code?

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

Free Lists the Wrong Way

- Don’t forget, if you free memory, you can’t use it!

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

Free List the Correct Way

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

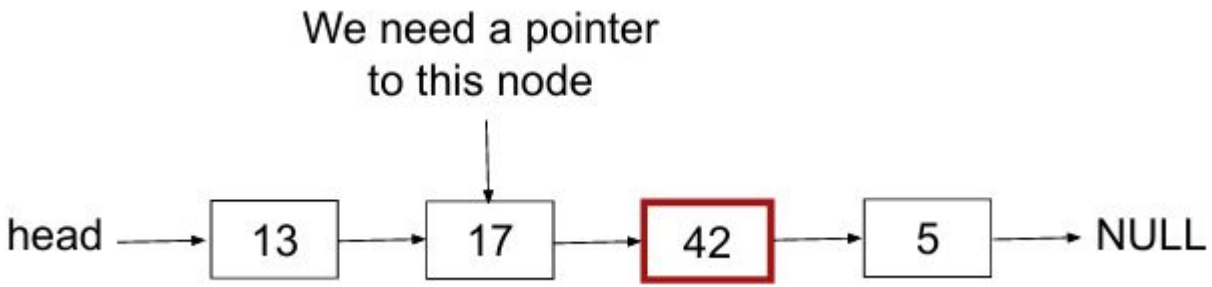
```c
// Delete all nodes from a given list
void free_list(struct node *head) {
    struct node *current = head;
    while (current != NULL) {
        head = head->next;
        free(current);
        current = head;
    }
}
```

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

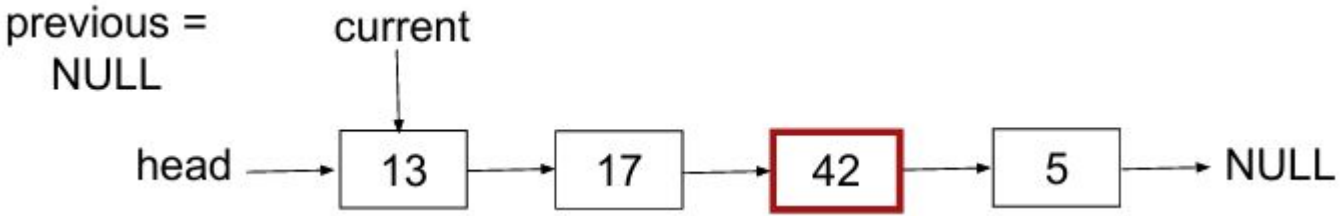
Search and Delete: Between Two Nodes

- To delete a node we need to link the previous node to the next node
  - E.g. if we want to delete the node with 42, we need to find the node before it



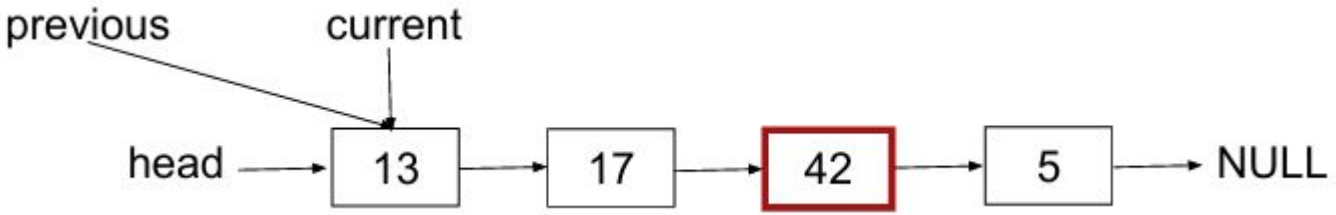
Search and Delete: Between Two Nodes

```
// 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;
}
```



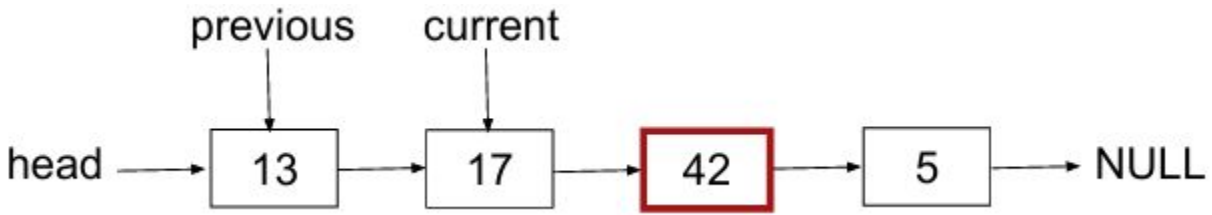
Search and Delete: Between Two Nodes

```
// 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;
}
```



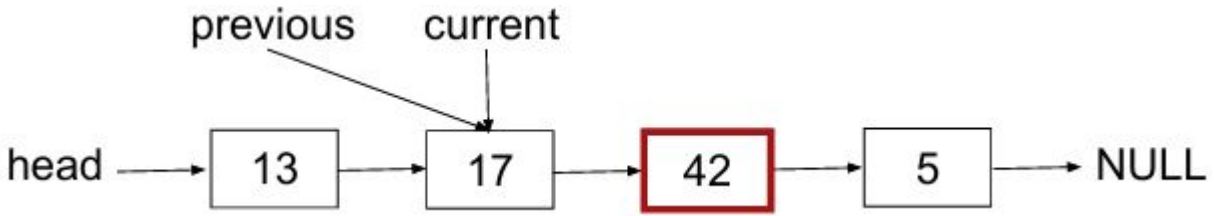
Search and Delete: Between Two Nodes

```
// 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;
}
```



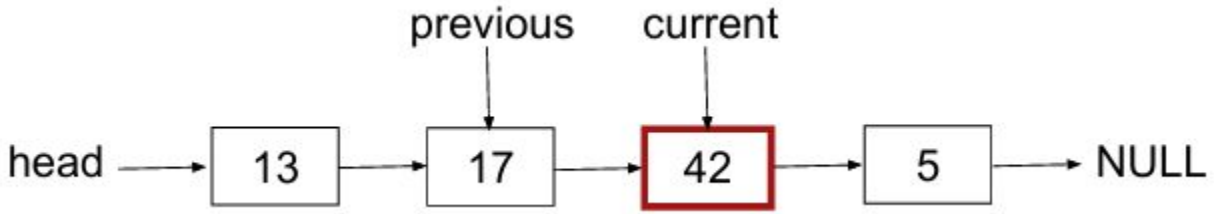
Search and Delete: Between Two Nodes

```
// 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;
}
```



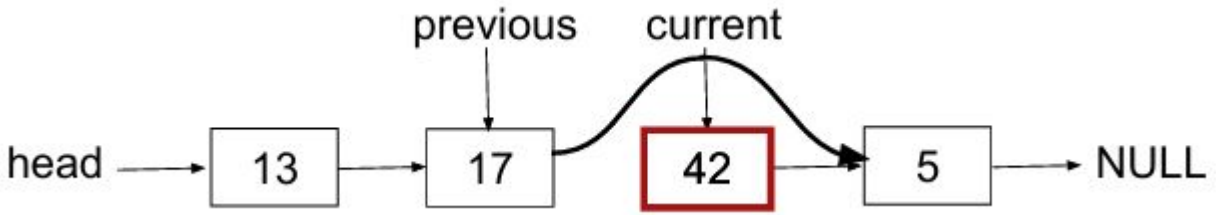
Search and Delete: Between Two Nodes

```
// 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

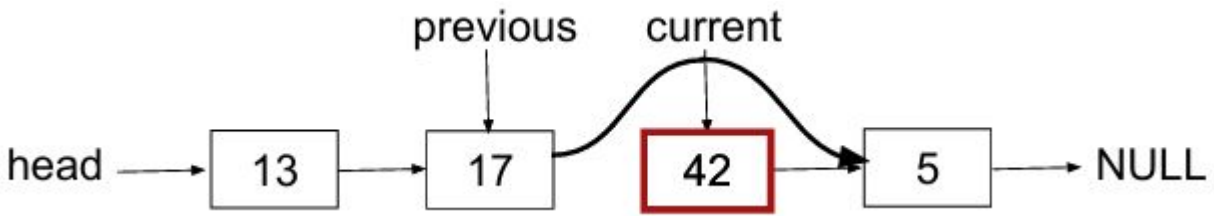
- Then we need to connect previous node to the one after the one we are deleting



Search and Delete Approach 1

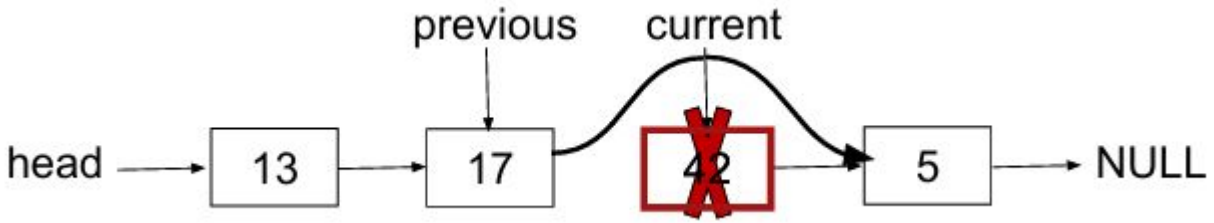
- Then we need to connect previous node to the one after the one we are deleting

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



Search and Delete Approach 1

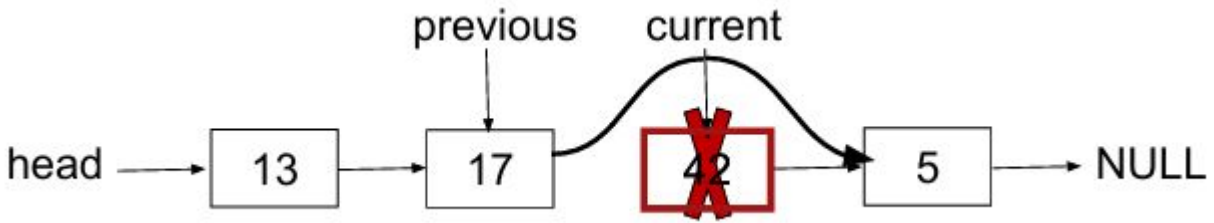
- Now we can free the node we want to delete



Search and Delete Approach 1

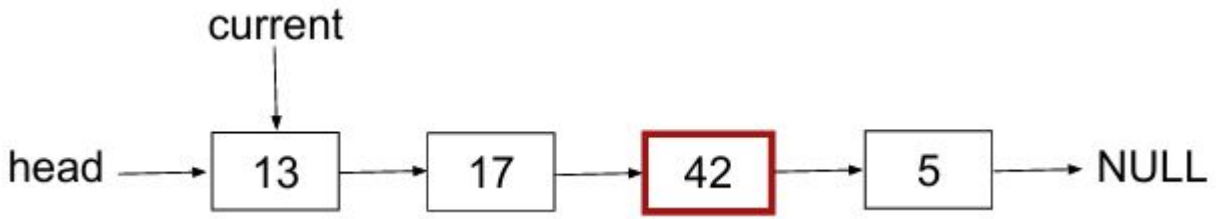
- Now we can free the node we want to delete

```
free(current);
```



Search and Delete Approach 2

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



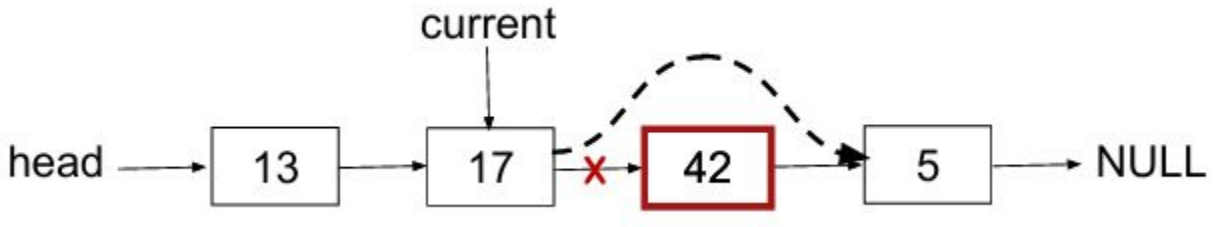
Search and Delete Approach 2: General Case

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



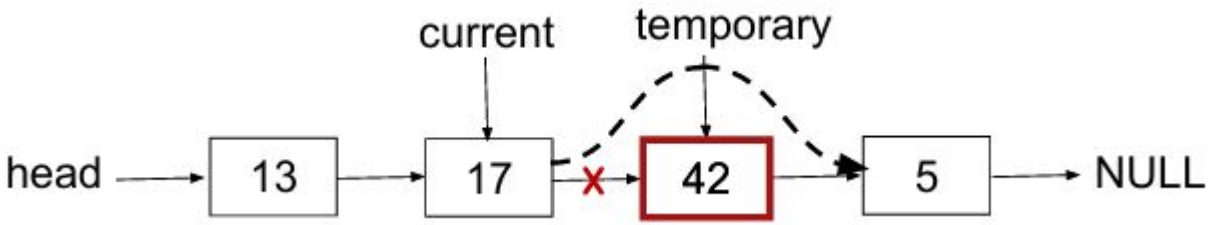
Search and Delete Approach 2

- Then we need to connect current node to the one after the one we're deleting
  - But we still need a pointer to the node we want to free, how can we do this?



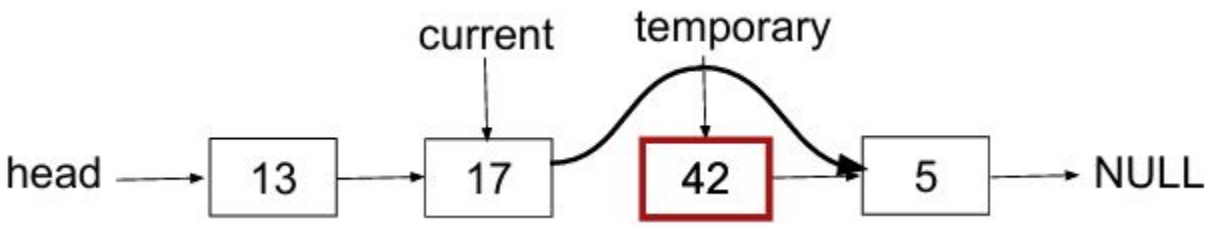
Search and Delete: Approach 2

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



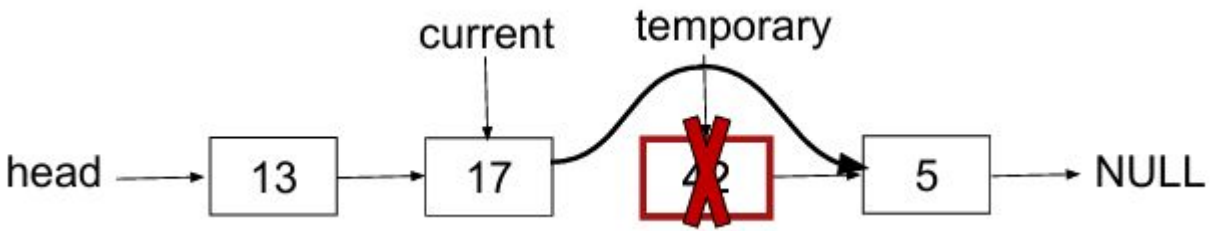
Search and Delete: Approach 2

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



Search and Delete: Approach 2

- Now we can free the node we want to delete

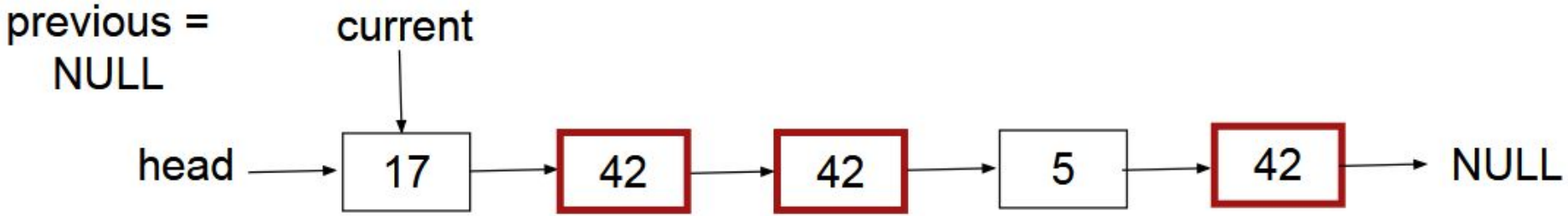


Coding

- Let’s code up these approaches

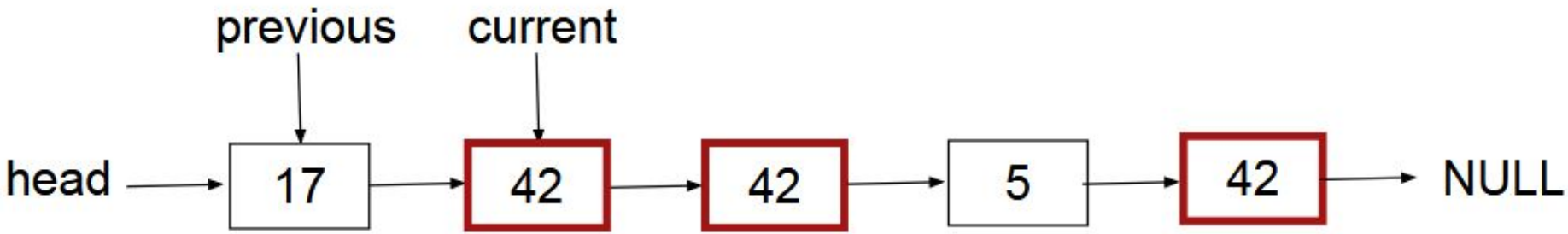
Search and Delete All

- Modify our search to delete all occurrences



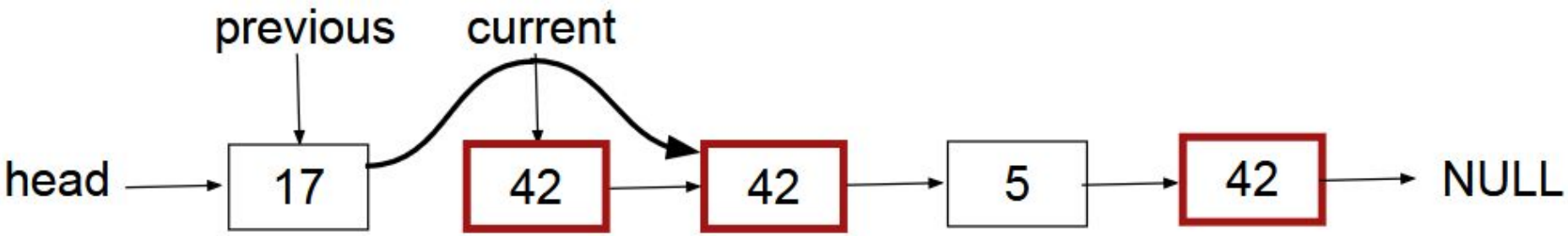
Search and Delete All

- Modify our search to delete all occurrences



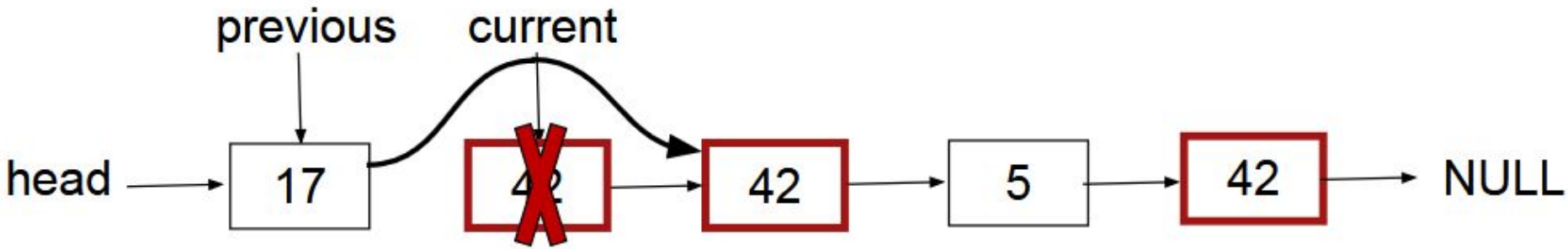
Search and Delete All

- Modify our search to delete all occurrences



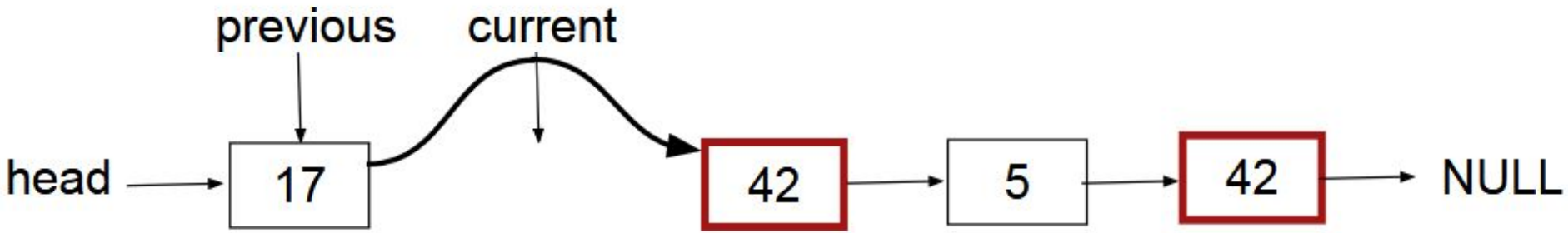
Search and Delete All

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



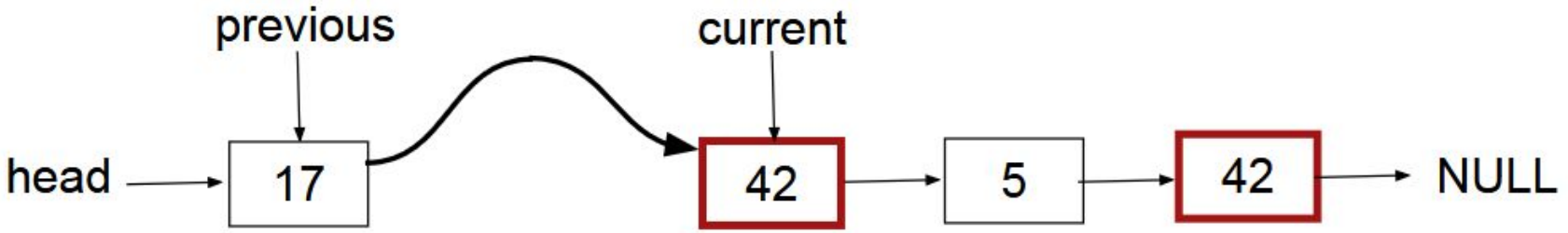
Search and Delete All

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



Search and Delete All

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




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

Break Time

...probably

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

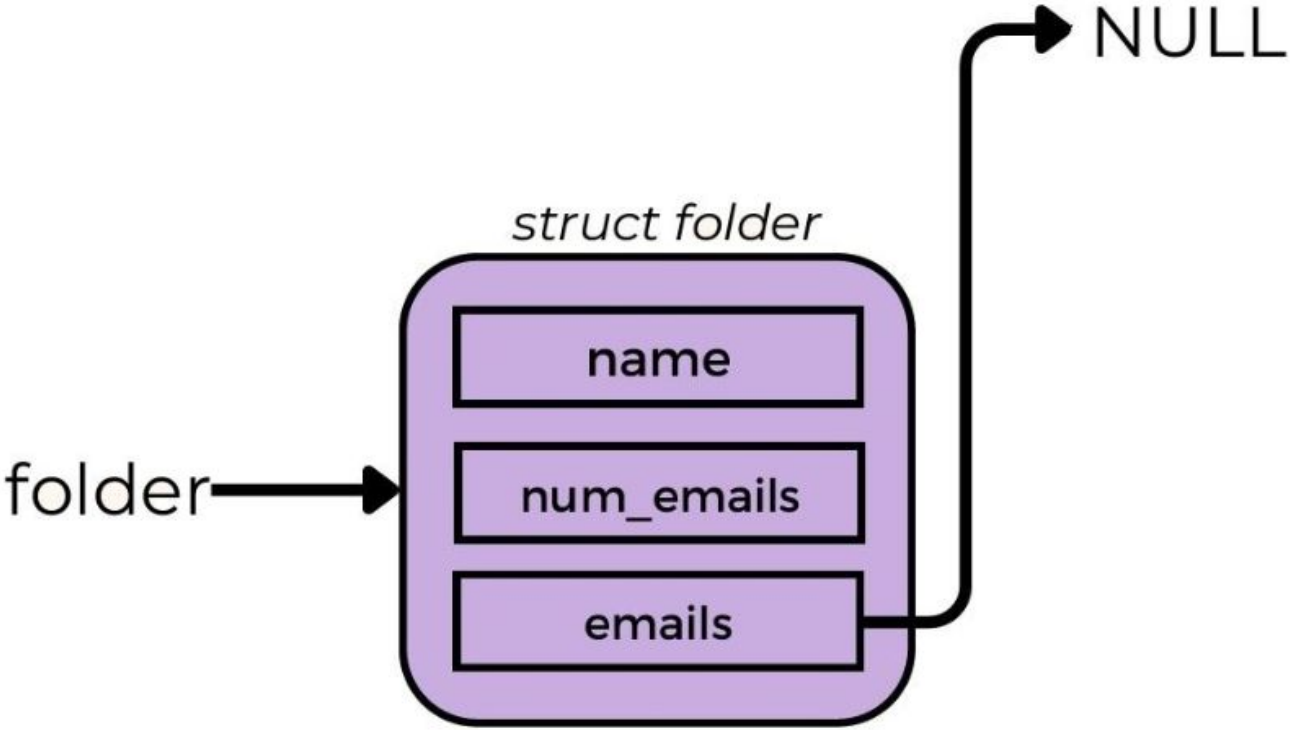
Structs

- Where are our nodes?
- Where is the head of the list?

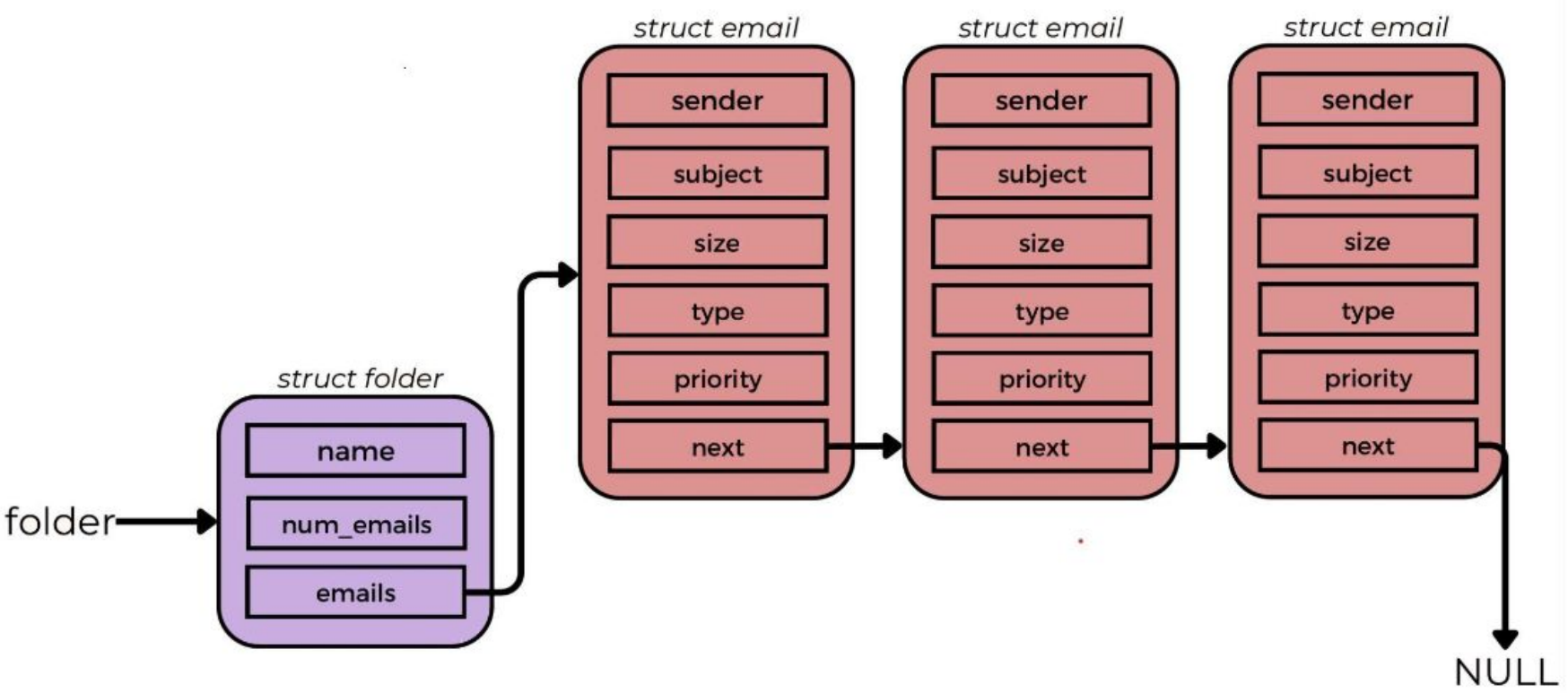
```
struct folder {
    char name[MAX_LEN];
    //to use later :)
    //int num_emails;
    struct email *emails;
};
```

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

Visualisation of the System

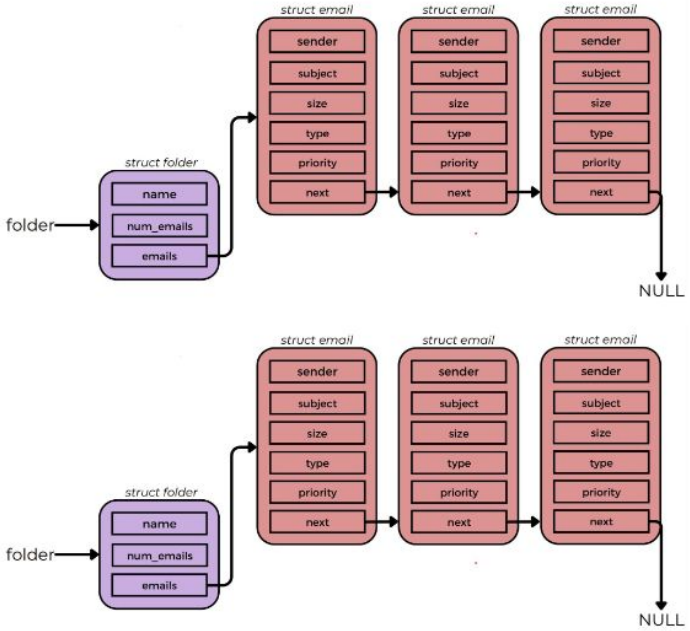


Visualisation of the System



Visualisation of the System

- We create many folders
  - Each containing a linked list 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 code program as follows

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 code program as follows

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

./test_main
```

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

./main
```

Functions to Write

- Stage 1
  - `create_folder`
  - `insert_email_at_head`
  - `search_emai`
- Stage 2
  - `clear_folders`
  - `delete_email_of_priority`
- Stage 3
  - `merge_folders`
  - `split_folder`

Stage 4 Extension

- 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 We Learnt

- 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 Time...

- Ange is back!!!!!
- Enjoy the rest of the course and your knowledge of linked lists!

Next Time

- More searching and deleting
  - Deleting from anywhere in a linked list recapped/continued
- Linked Lists - A Larger Application
  - Linked lists as members in other structs
  - Linked lists with more complex data types
  - Helpful for Assignment 2!

Reach Out

- Check the course forum for questions!
  - Post there too!
- Admin questions
  - cs1511@unsw.edu.au

Thank You

Questions?