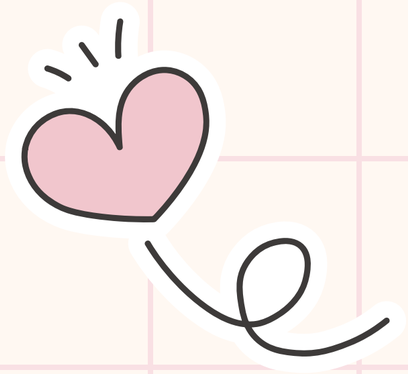
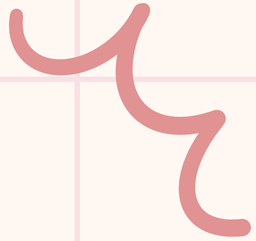
COMP1511 Programming Fundamentals





MORE LINKED LISTS!
(CODING + DRAWING TIME)

WEEK 7 LECTURE 2

with Tammy

(for Wk7 Mon - Wk9 Mon)

(Sasha is at a conference across the globe)

Course Admin

Tammy Zhong

She/Her

Announcements

ASSIGNMENT 2

RELEASING TOMORROW

ALL ABOUT LINKED LISTS

START EARLY!

DUE END OF WEEK 10

ASSIGNMENT 2 LIVESTREAM

NEXT TUESDAY

1:00PM

113 SEMINAR RM K17

+ YOUTUBE

We will try and get assignment 1 marks and style feedback released before assignment 2's deadline so you can learn from it to apply to your assignment 2

Announcements

EASTER CATCH UP CLASSES

IF YOUR TUT-LAB FALLS ON GOOD FRIDAY OR EASTER MONDAY

SIGN UP VIA LINK ON FORUM!

WEEK 8 REVISION SESSIONS

SIGN UP LINK ON COURSE FORUM!

LIVE CODE HERE:

https://cgi.cse.unsw.edu.au/~cs1511/24T1/live/week_7/

THIS WEEK:
INTO THE WORLD OF LINKED LISTS

Last Lecture:

- Pointers & Memory Recap
- Linked Lists
  - conceptual introduction
  - insert at head (we got up to here)
  - traverse a linked list
  - insert at tail

THIS WEEK:
INTO THE WORLD OF LINKED LISTS

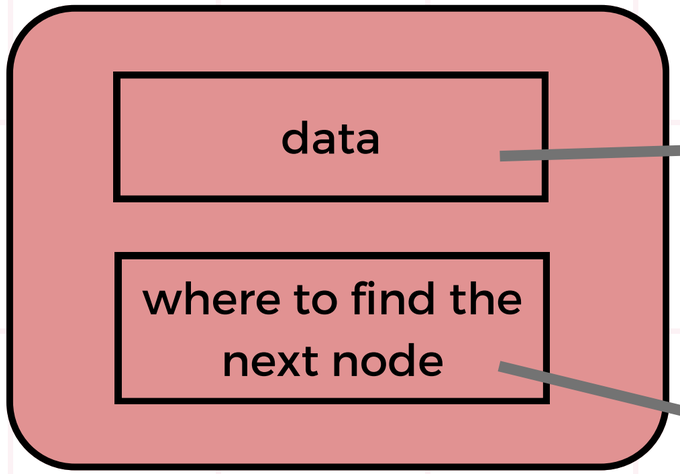
Today - pick up from where we left off:

- More linked lists!
  - insert at head (continue from last lecture)
  - traverse a linked list
  - insert at tail
  - insert anywhere
  - intro to deleting nodes (maybe)

Quick Recap:
What does a linked list look like again?

A “node” in a linked list:

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

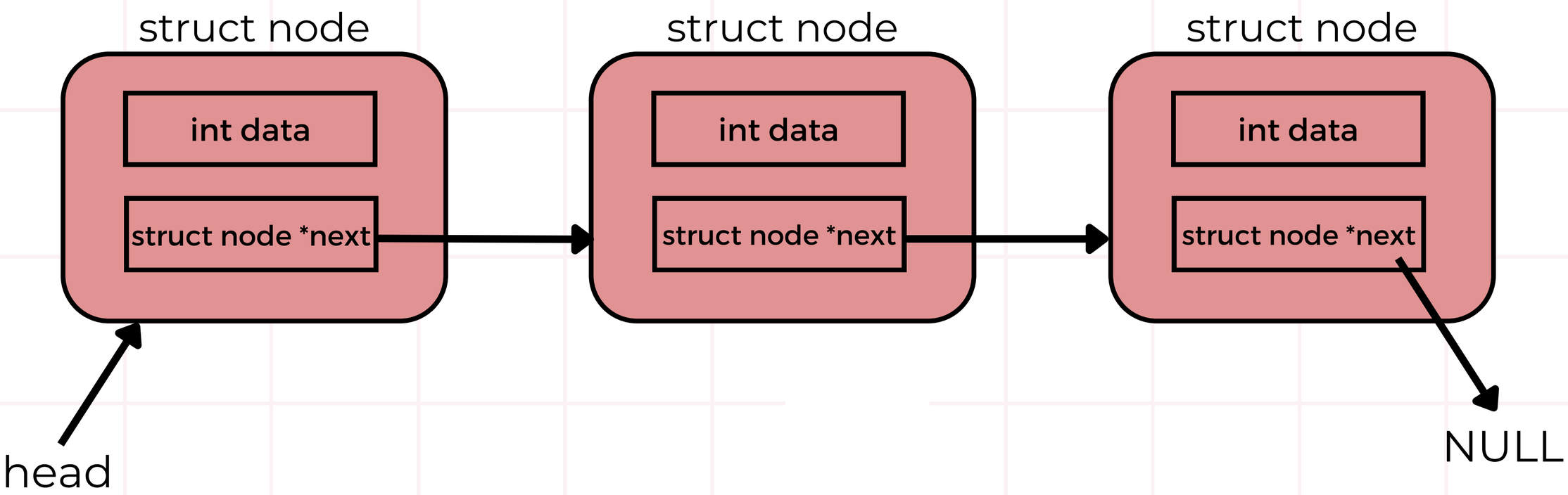


An integer variable - to store the number
e.g. 7

A variable storing the address of the next node (i.e. a pointer) - so there's a way to connect nodes together
e.g. 0x66

LINKED LIST

Example:

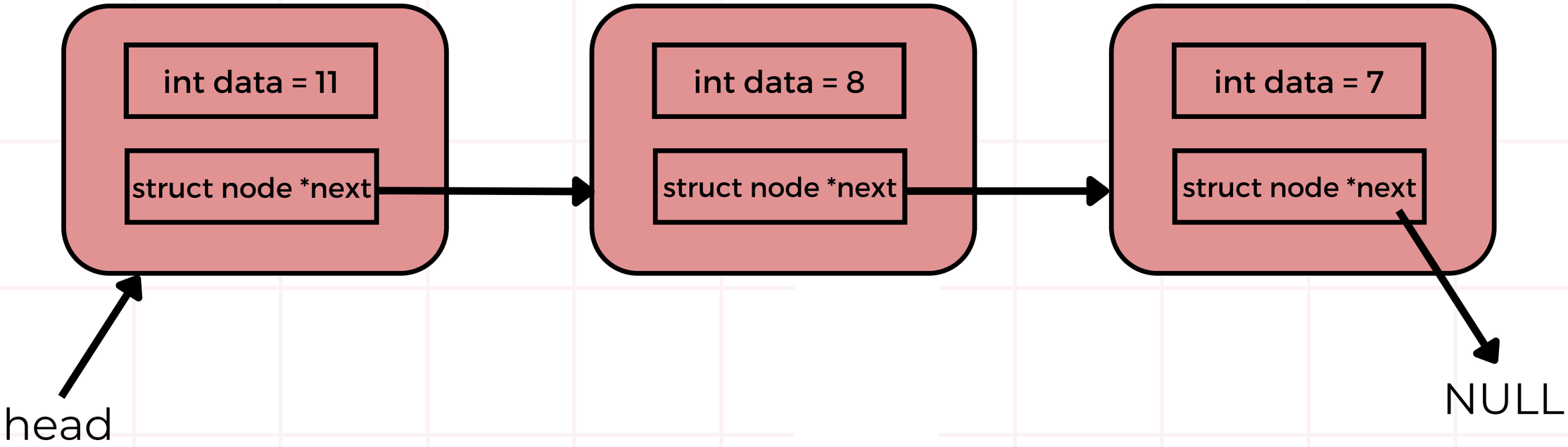


Note: You can have more than `int data` for each element/"node"!

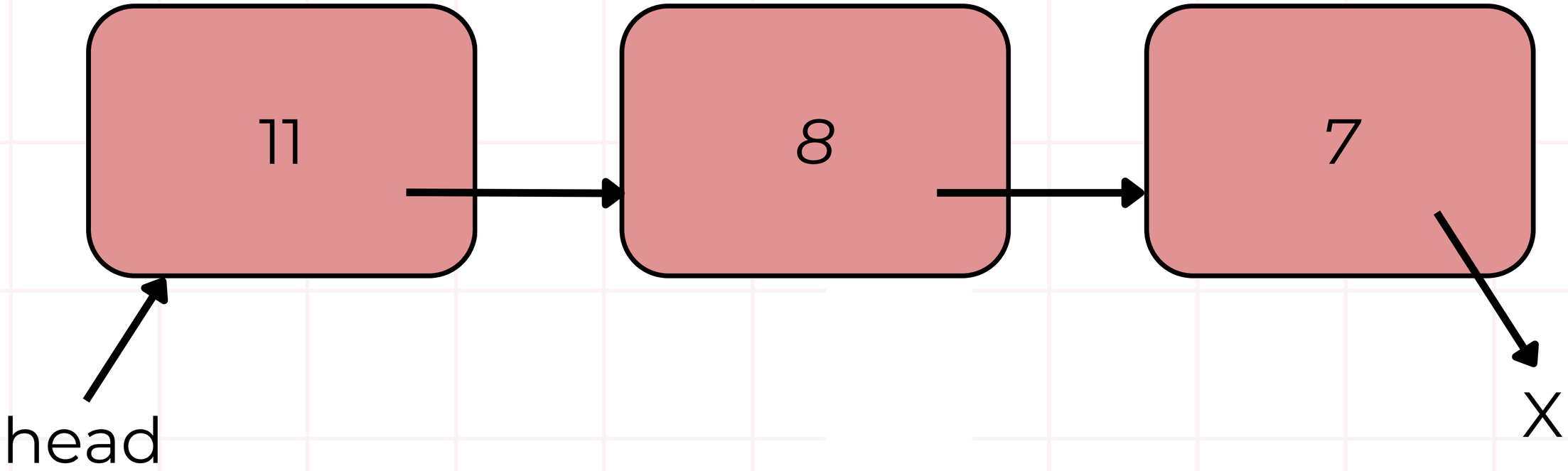
LINKED LISTS!

An Empty Linked List looks like:

From now on, if a linked list looks like this:



I will draw them like this (to make it easier to draw):



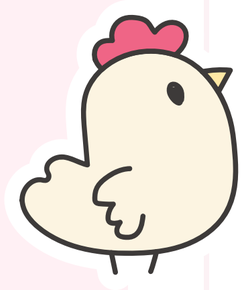
Our unfinished task from last lecture...

create a linked list to store the numbers ***11, 8 , 7***
as elements

LET'S PUT THIS LINKED LIST TOGETHER IN CODE (W/ ACTUAL DRAWINGS)!

11->8->7->X

We will need to know how to use struct pointers and malloc!



Steps to do this!

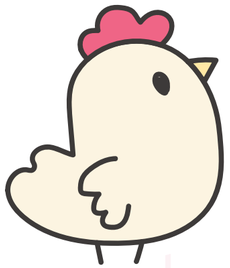
1. Define a struct for our node
2. Declare a pointer to keep track of the beginning of list
3. Code to create a node and connect it to a linked list

DIAGRAMS!

DIAGRAMS!

CODING TIME!

11->8->7->X



WHAT DID WE JUST CODE UP?

- *Created a linked list by inserting nodes at the head*
- *We are inserting backwards; last element inserted first*

HOW DO WE INSERT "FORWARD"?

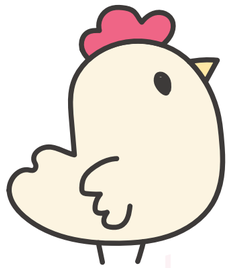
- *need to insert at tail*
- *need to know how to traverse the linked list to get to the end to do so*

DIAGRAMS!

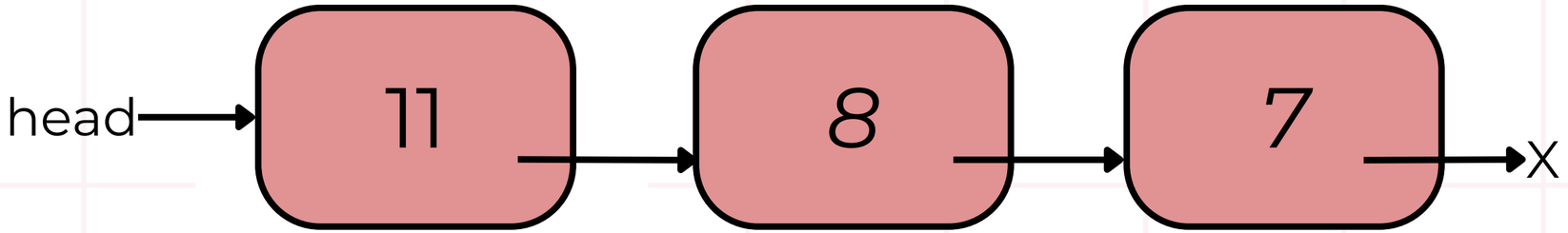
DIAGRAMS!

CODING TIME! (AGAIN)

Traverse the linked list and print the data!

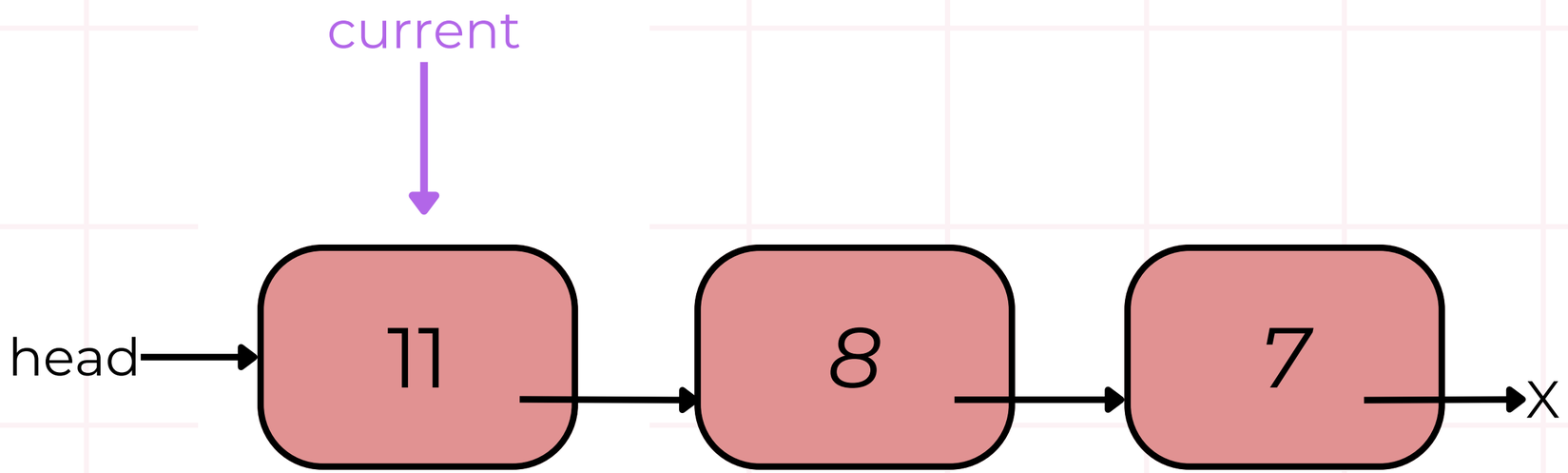


TRAVERSING A LINKED LIST



TRAVERSING A LINKED LIST

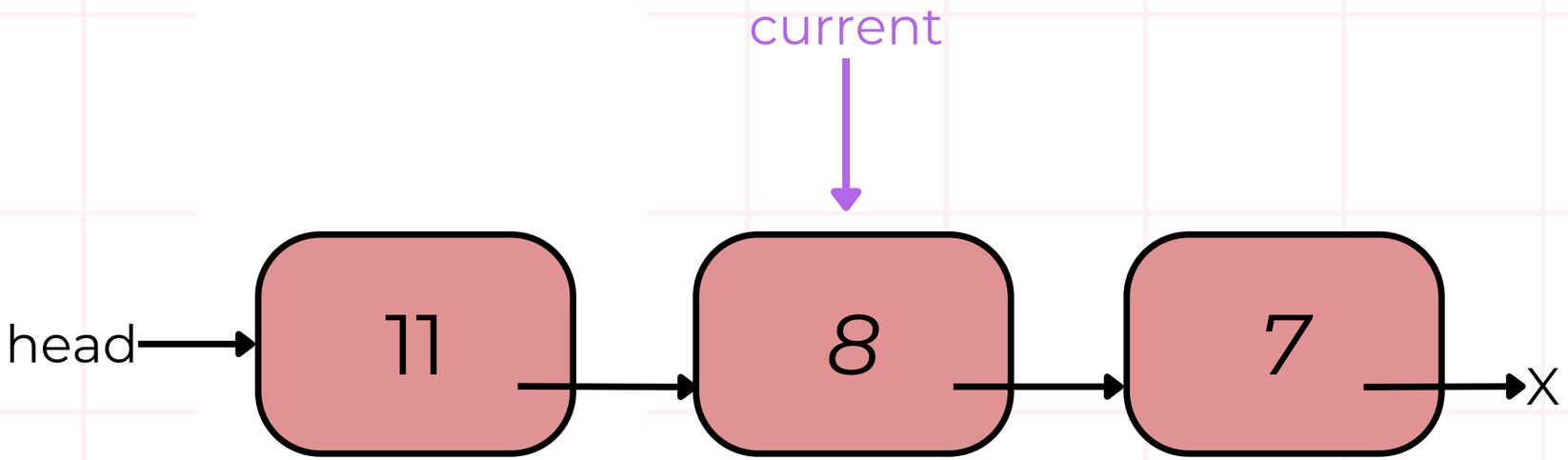
*struct node *current = head;*



TRAVERSING A LINKED LIST

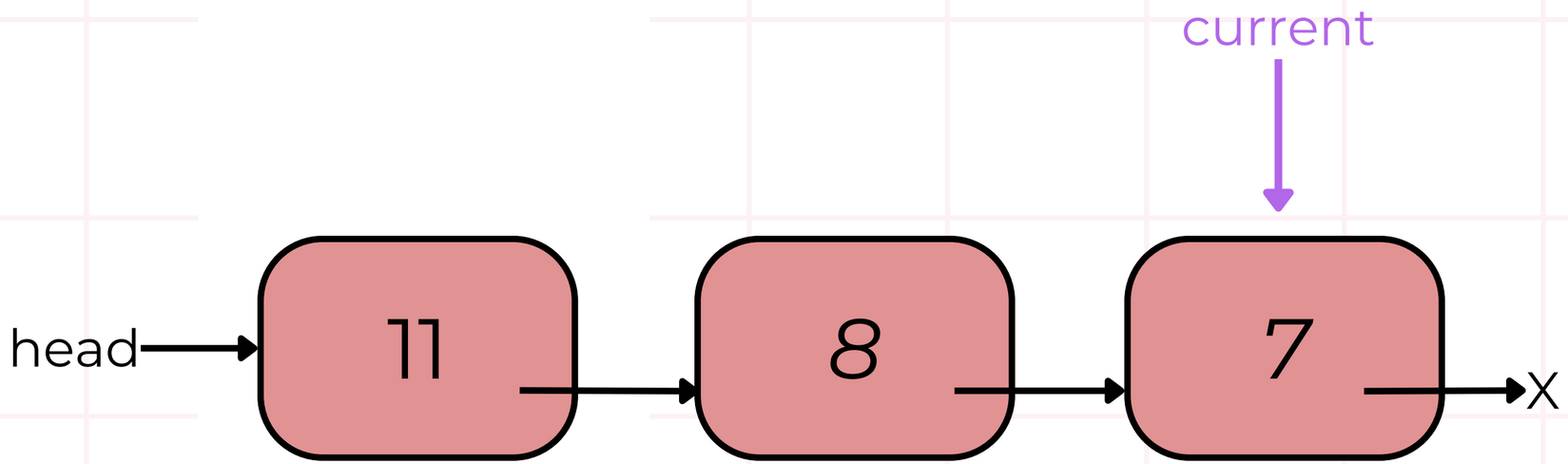
*struct node *current = head;*

current = current->next;



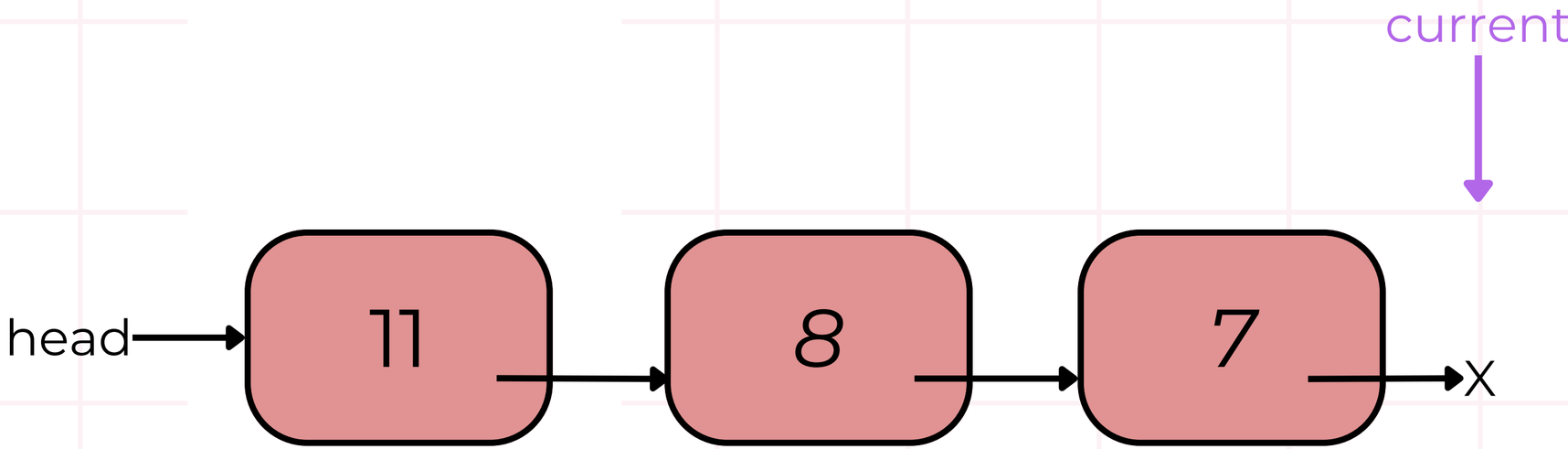
TRAVERSING A LINKED LIST

*struct node *current = head;*
current = current->next;
current = current->next;



TRAVERSING A LINKED LIST

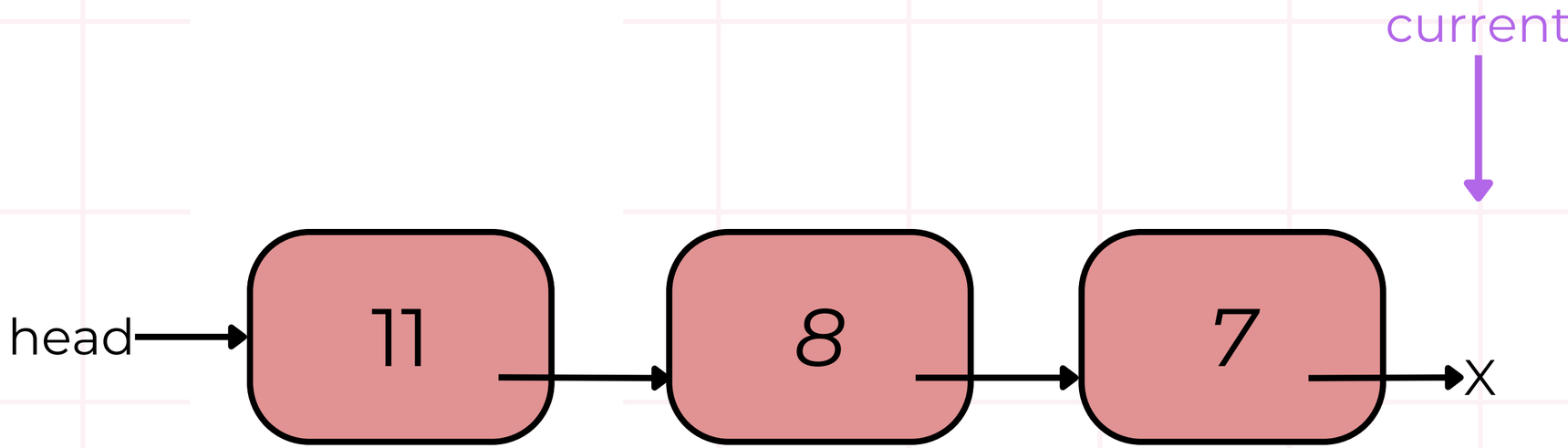
*struct node *current = head;*
current = current->next;
current = current->next;
current = current->next;



TRAVERSING A LINKED LIST

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

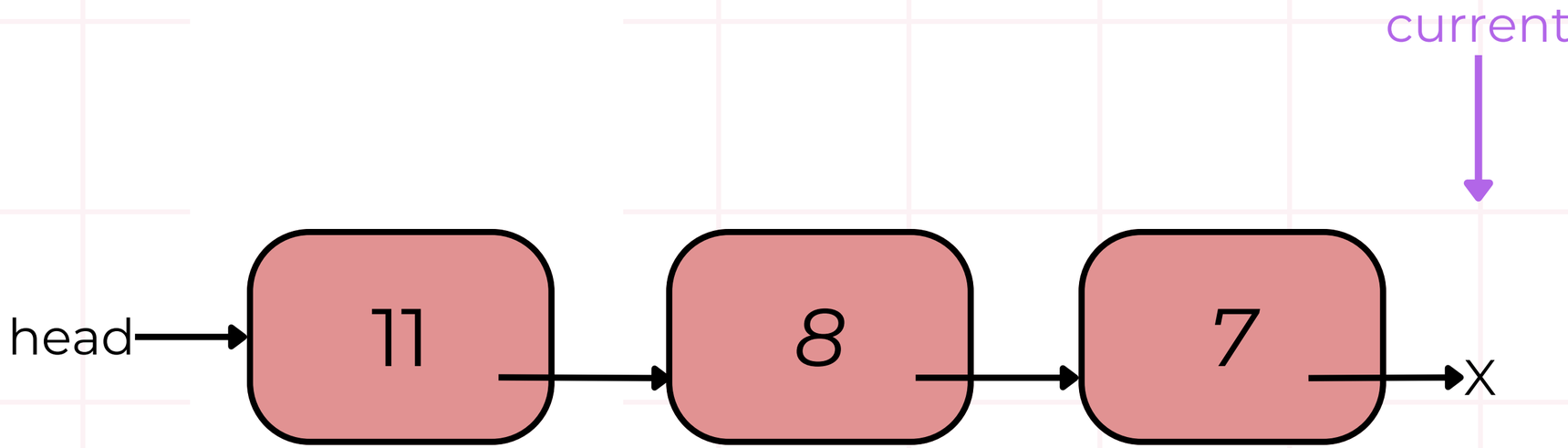
Repetition of code means it's time to use a loop!



TRAVERSING A LINKED LIST

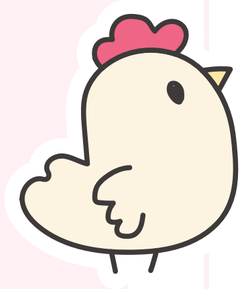
```
struct node *current = head;
while (current != NULL) {
    // ...
    current = current->next;
}
```

Repetition of code means it's time to use a loop!



HOW DO WE INSERT A NEW NODE WITH THE VALUE OF 6 AT THE TAIL OF THIS LIST?

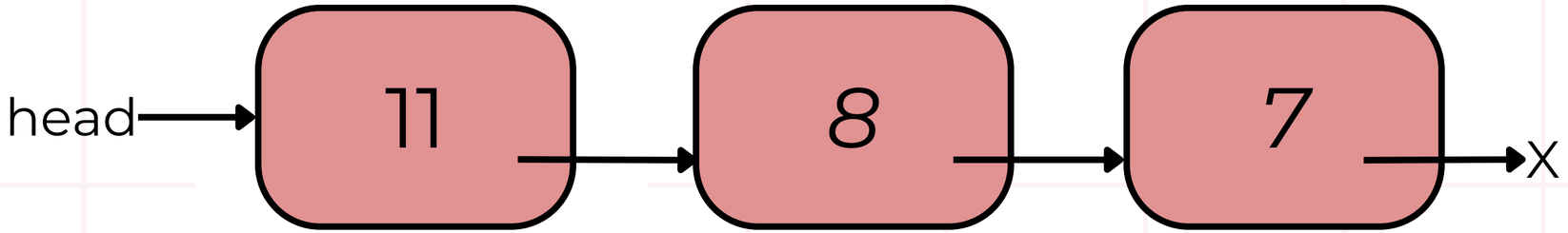
11->8->7->X



Steps to do this!

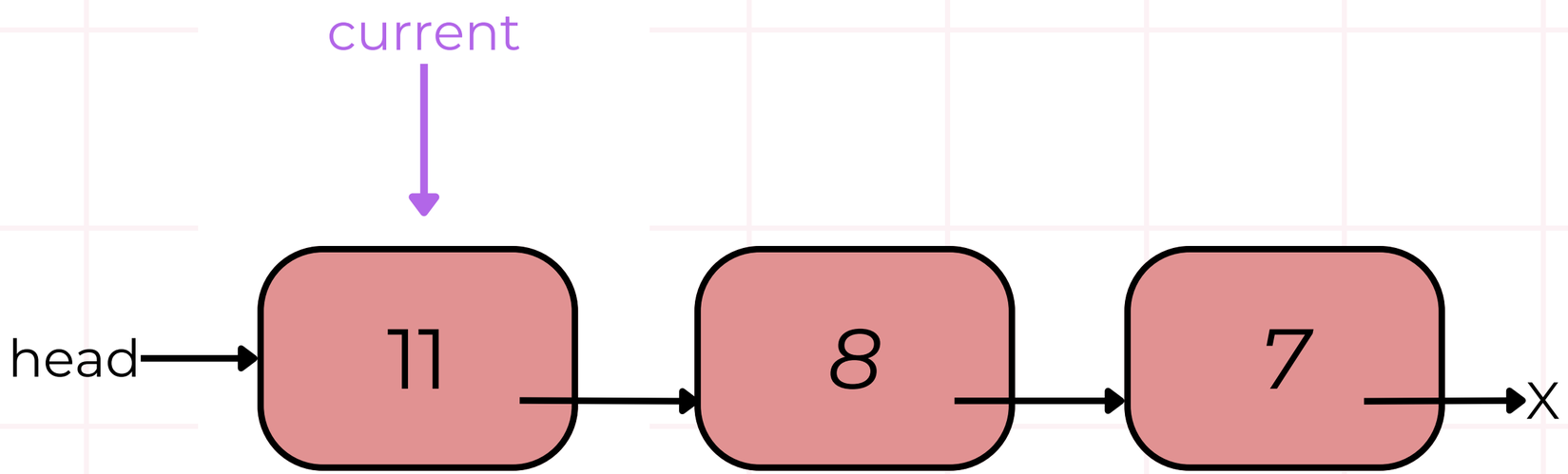
1. Malloc memory for a new node and initialise it
2. Traverse to the last node in the given list
3. Add the new node after this current last node of the list

INSERT AT TAIL



INSERT AT TAIL

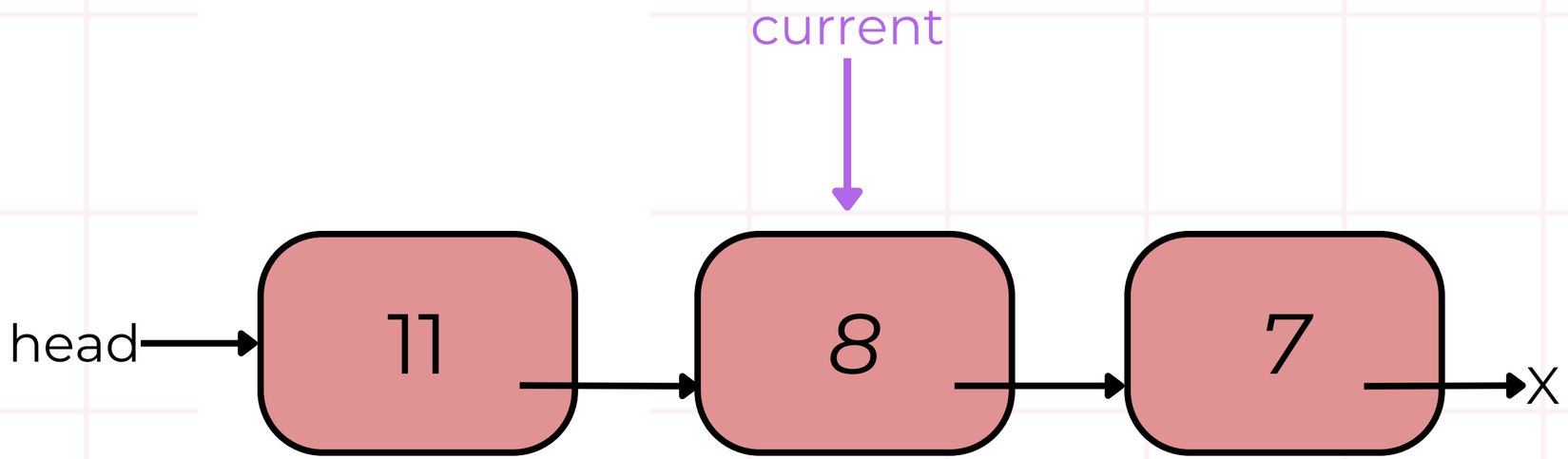
*struct node *current = head;*



INSERT AT TAIL

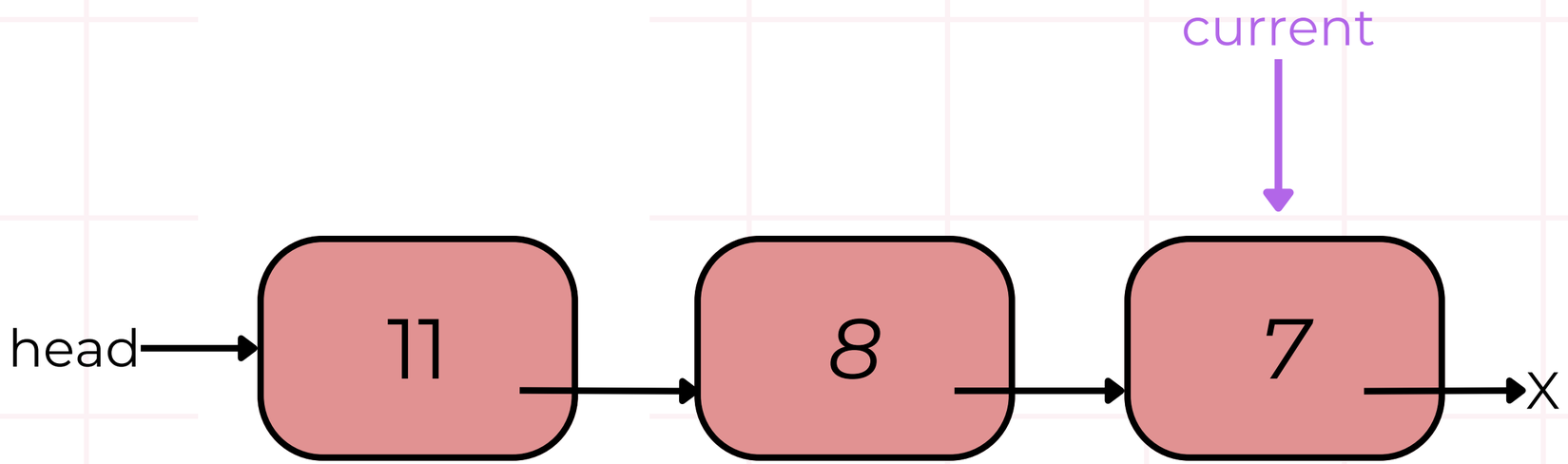
*struct node *current = head;*

current = current->next;



INSERT AT TAIL

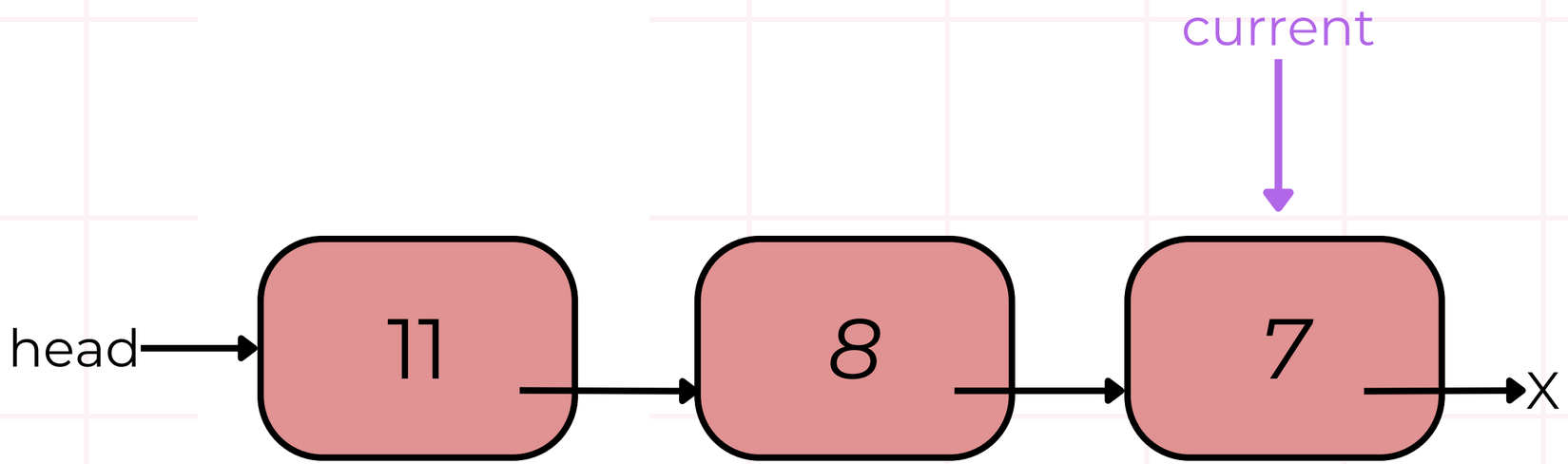
*struct node *current = head;*
current = current->next;
current = current->next;



INSERT AT TAIL

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

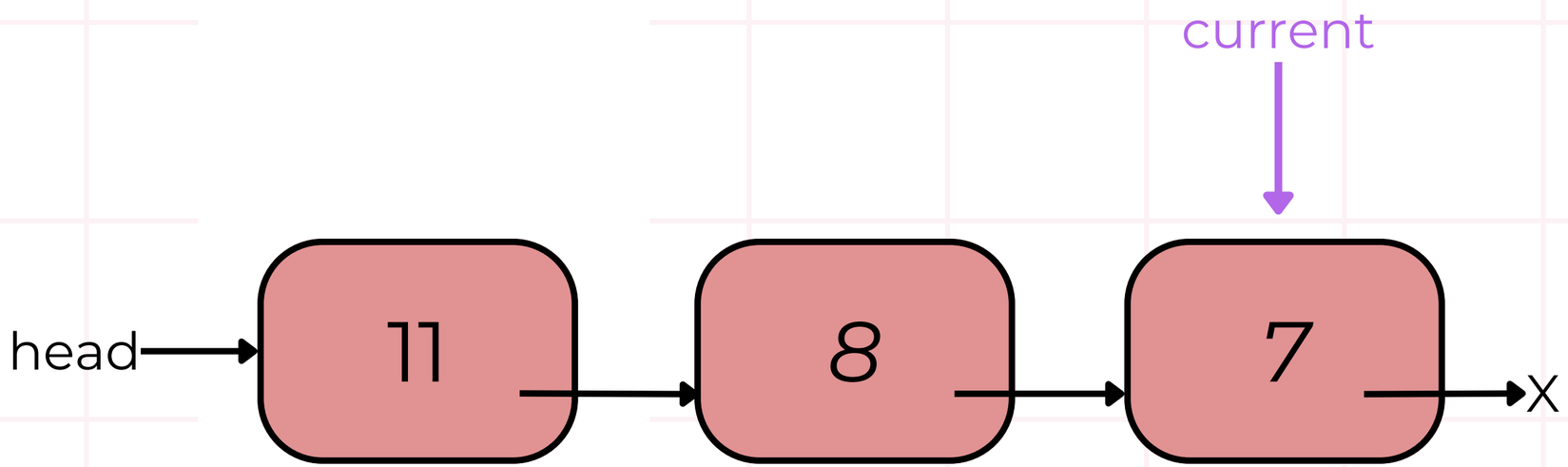
Again, repetition means a loop!



INSERT AT TAIL

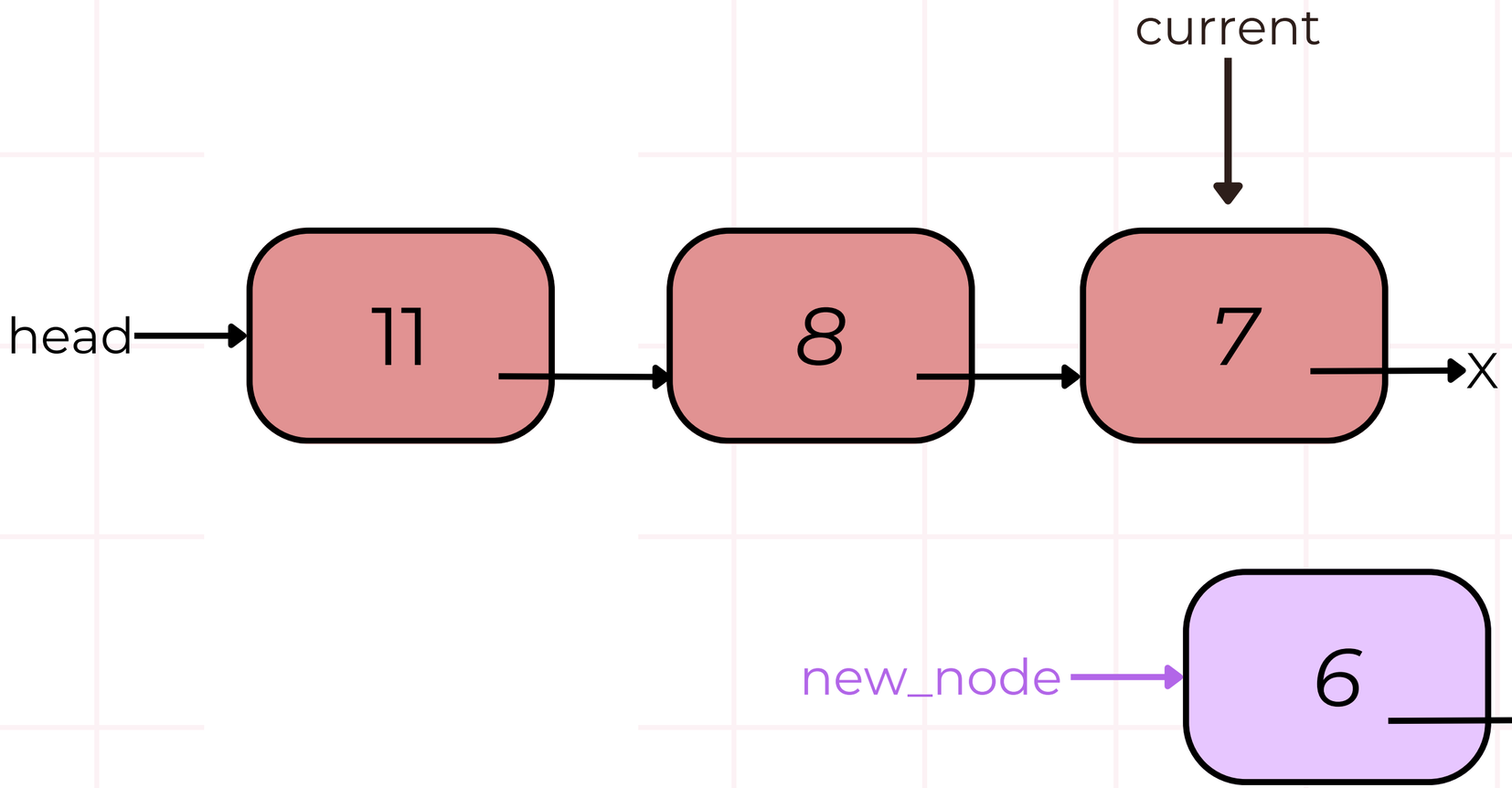
```
struct node *current = head;
while (current->next != NULL) {
    current = current->next;
}
```

Again, repetition means a loop!



INSERT AT TAIL

```
struct node *current = head;
while (current->next != NULL) {
    current = current->next;
}
```

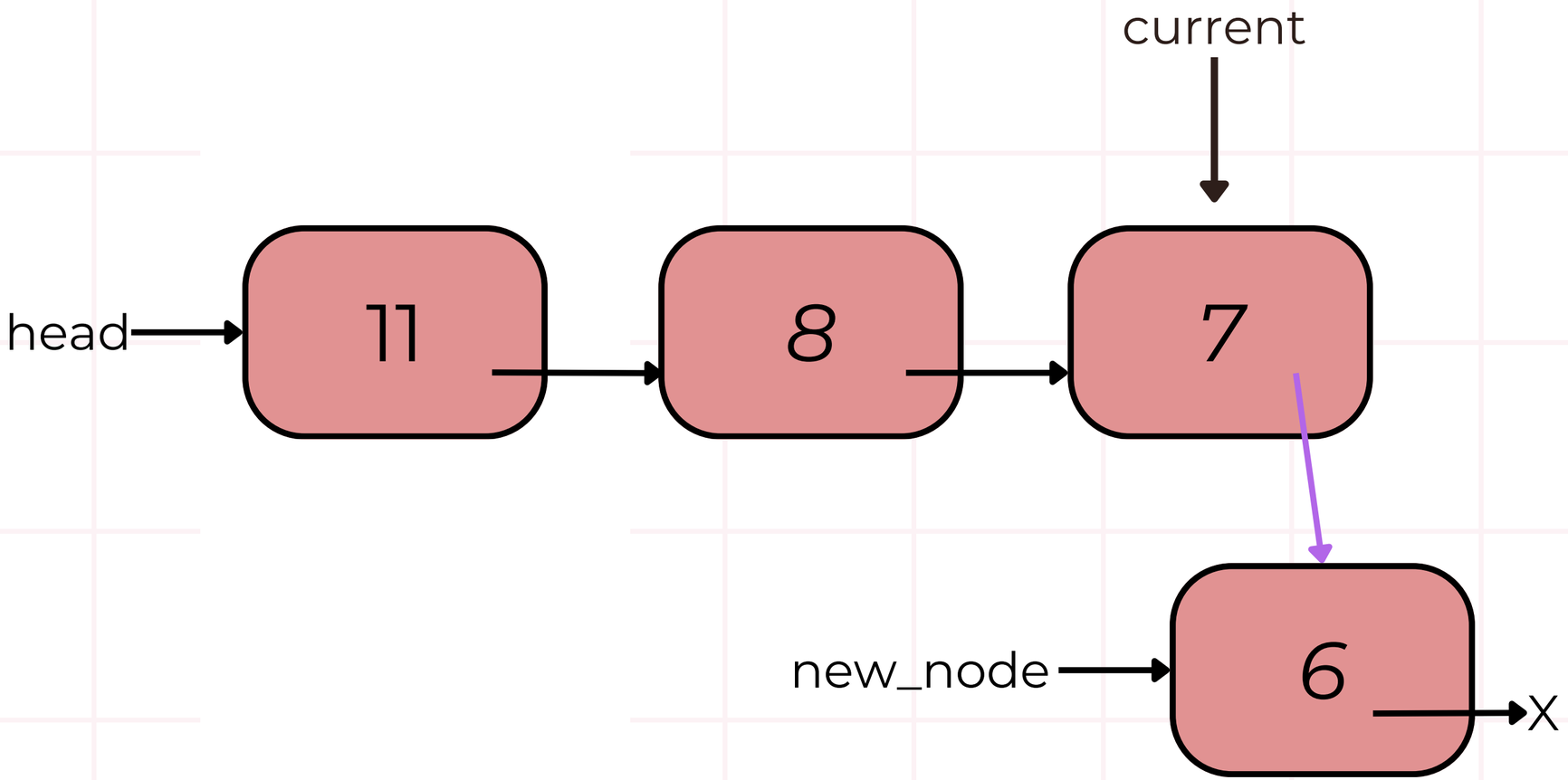


Let's assume we have a new_node somewhere for now (this will need to be malloc-ed beforehand)

INSERT AT TAIL

```
struct node *current = head;
while (current->next != NULL) {
    current = current->next;
}
```

current->next = new_node

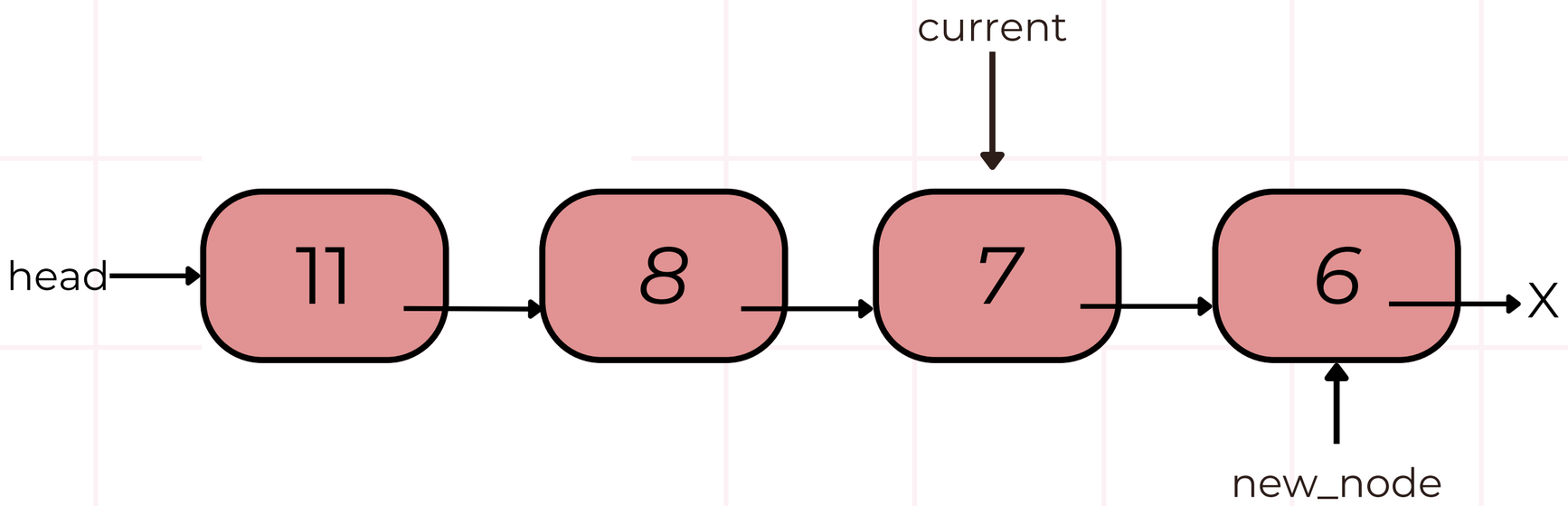


Note: this is incomplete pseudocode

INSERT AT TAIL

```
struct node *current = head;
while (current->next != NULL) {
    current = current->next;
}

current->next = new_node
```



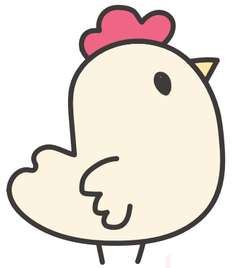
Note: this is incomplete pseudocode

DIAGRAMS!

DIAGRAMS!

CODING TIME! (AGAIN)

Insert at Tail



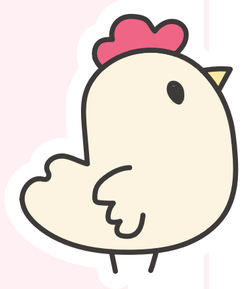
WAIT, ARE WE DONE WITH INSERTION AT TAIL?

That is,

- *Will our code work fine for different "edge cases"?*
  - *e.g. inserting at tail in an empty list...*

BREAK TIME!

GIVEN A LIST, AN INTEGER VALUE AND A POSITION NUMBER, HOW DO WE INSERT A NEW NODE WITH THE INTEGER VALUE AFTER THE SPECIFIED POSITION?



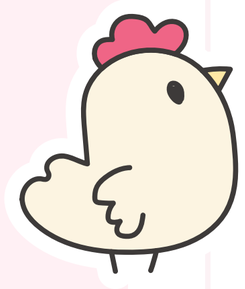
INSERT ANYWEHRE (AT SPECIFIED POSITION)

For example, given:

- the list,
  - 11->8->7->6->X
- an integer value of
  - 5
- a position number of
  - 2

The list will become:

- 11->8->**5**->7->6->X



- i*f position number == 0, insert as the first element of the list*
- *if position number > the number of elements in the list or < 0, do nothing)*

HOW DO WE INSERT ANYWHERE*
(NOT JUST HEAD OR TAIL)?

We need to consider some "cases":

- *Where in the list can we insert into:*
  - *at the head? (as the first node)*
  - *between two existing nodes in the list?*
  - *at the tail? (as the last node)*
- *How many nodes do we have in the list?*
  - *Empty list?*
  - *Only one node in the list?*
  - *More than one / many nodes in the list?*
- ...

**how do we write a function that would just insert an element wherever we specify*

General steps to do this!

1. Malloc memory for a new node and initialise it
2. Find the node right before the position to insert into
3. Change where pointers are pointing to to add the new node into the list accordingly

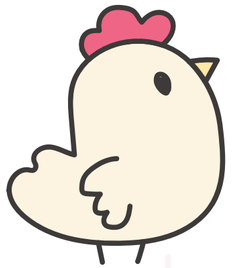
Don’t forget to also consider different edge cases

DIAGRAMS!

DIAGRAMS!

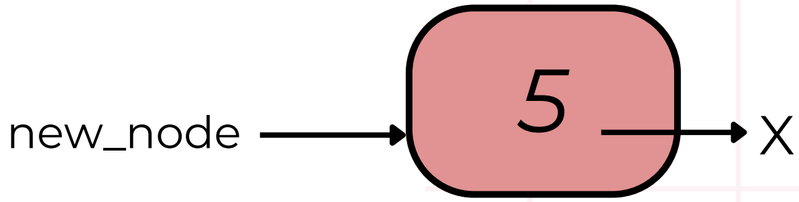
CODING TIME! (AGAIN)

Insert at position (anywhere)



INSERT ANYWEHRE

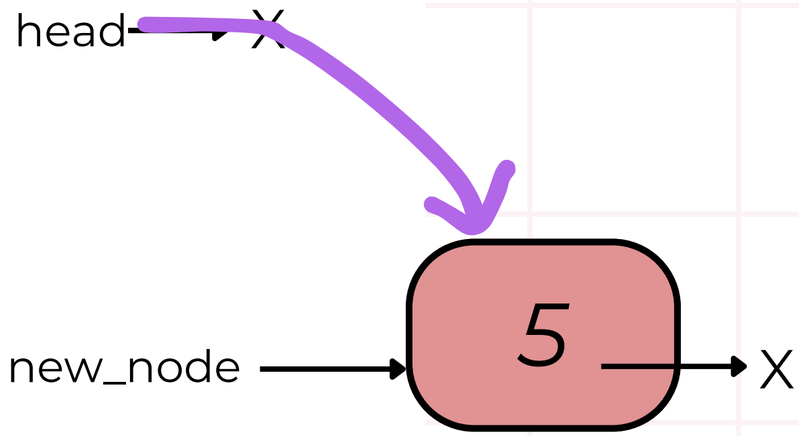
EXAMPLE CASE #1: INSERT NEW NODE AT THE HEAD (I.E. WHEN POSITION NUMBER == 0) OF AN EMPTY LIST



**for the purpose of having an example,*
we will explore inserting exactly in the middle of the above list

INSERT ANYWEHRE

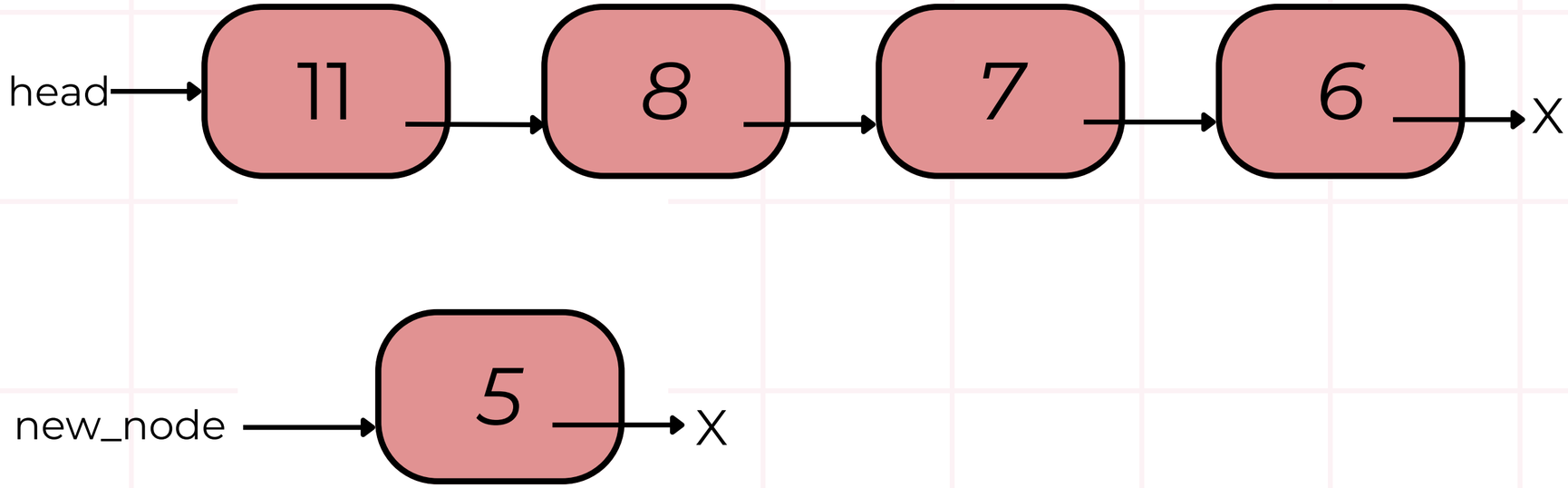
EXAMPLE CASE #1: INSERT NEW NODE AT THE HEAD (I.E. WHEN POSITION NUMBER == 0) OF AN EMPTY LIST



**for the purpose of having an example,*
we will explore inserting exactly in the middle of the above list

INSERT ANYWEHRE

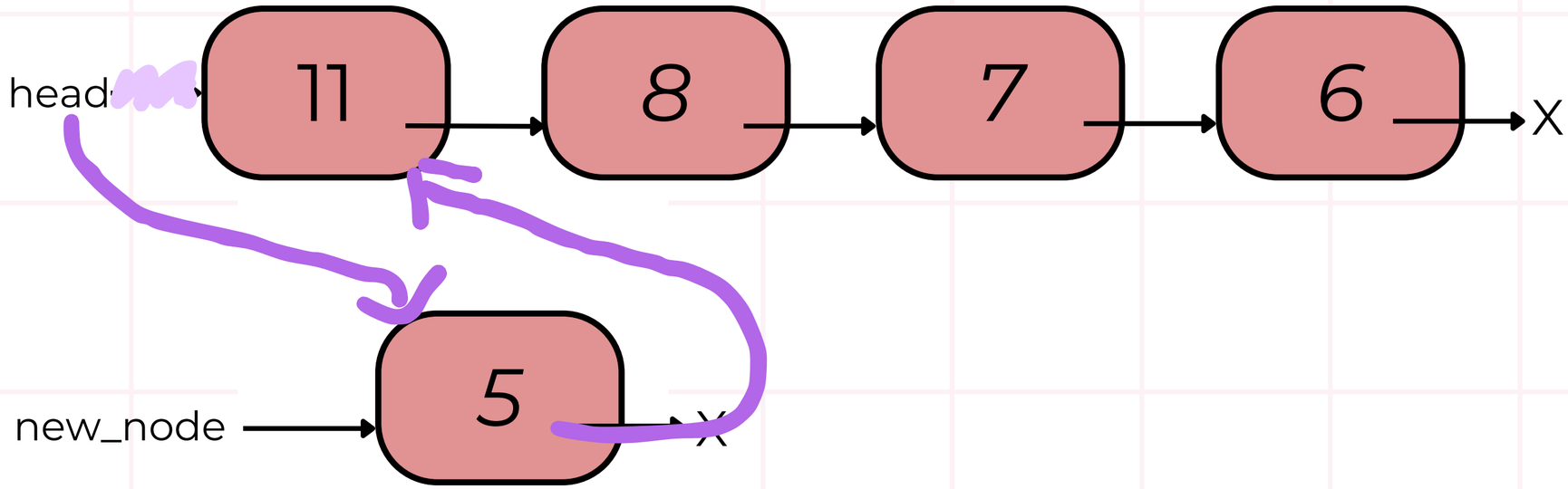
EXAMPLE CASE #2: INSERT NEW NODE AT THE HEAD (I.E. WHEN POSITION NUMBER == 0) OF A NON-EMPTY LIST



**for the purpose of having an example,*
we will explore inserting exactly in the middle of the above list

INSERT ANYWEHRE

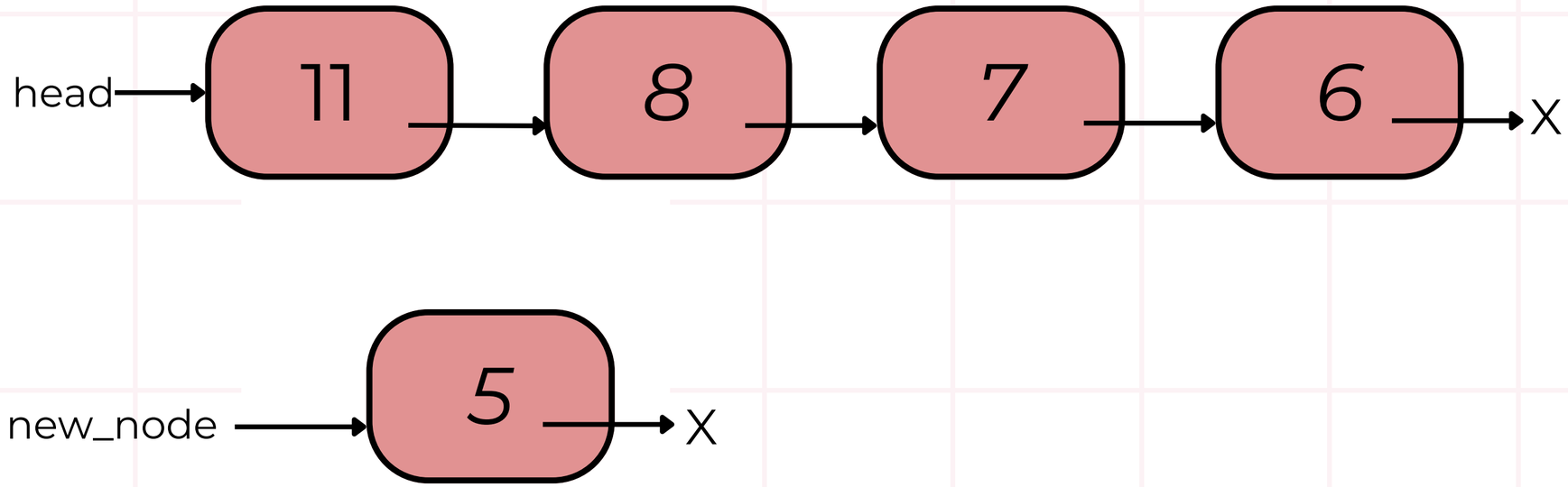
EXAMPLE CASE #2: INSERT NEW NODE AT THE HEAD (I.E. WHEN POSITION NUMBER == 0) OF A NON-EMPTY LIST



**for the purpose of having an example,*
we will explore inserting exactly in the middle of the above list

INSERT ANYWEHRE

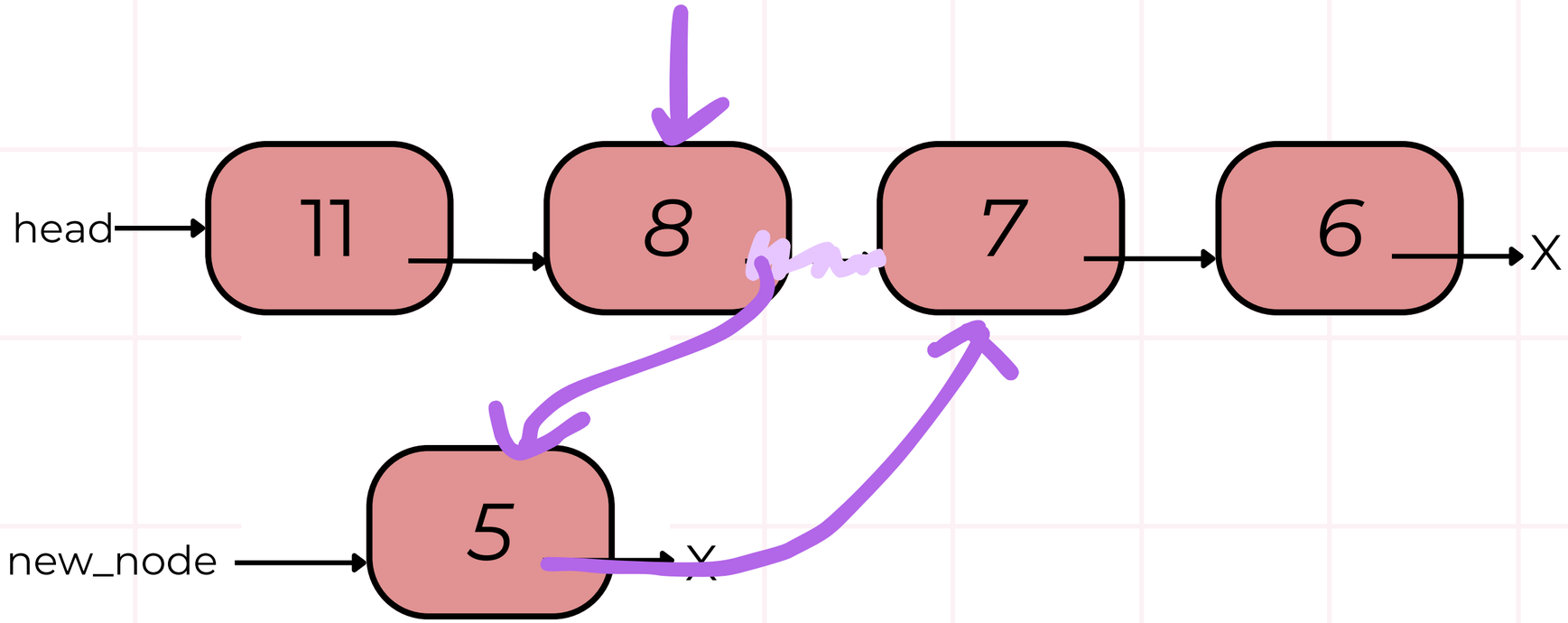
EXAMPLE CASE #3 (THE AVERAGE CASE):
INSERT NEW NODE
IN BETWEEN TWO NODES*
IN A LIST WITH MULTIPLE NODES



**for the purpose of having an example,*
we will explore inserting exactly in the middle of the above list

INSERT ANYWEHRE

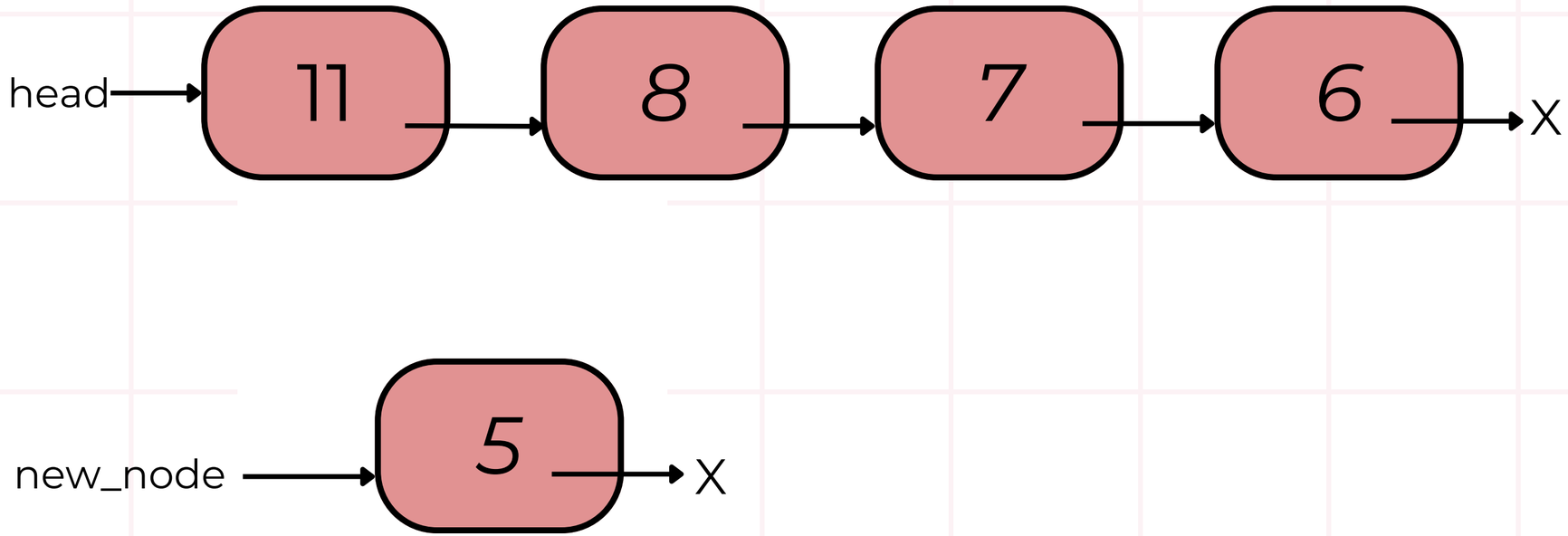
EXAMPLE CASE #3: INSERT NEW NODE IN BETWEEN TWO NODES* IN A LIST WITH MULTIPLE NODES



**for the purpose of having an example,*
we will explore inserting exactly in the middle of the above list

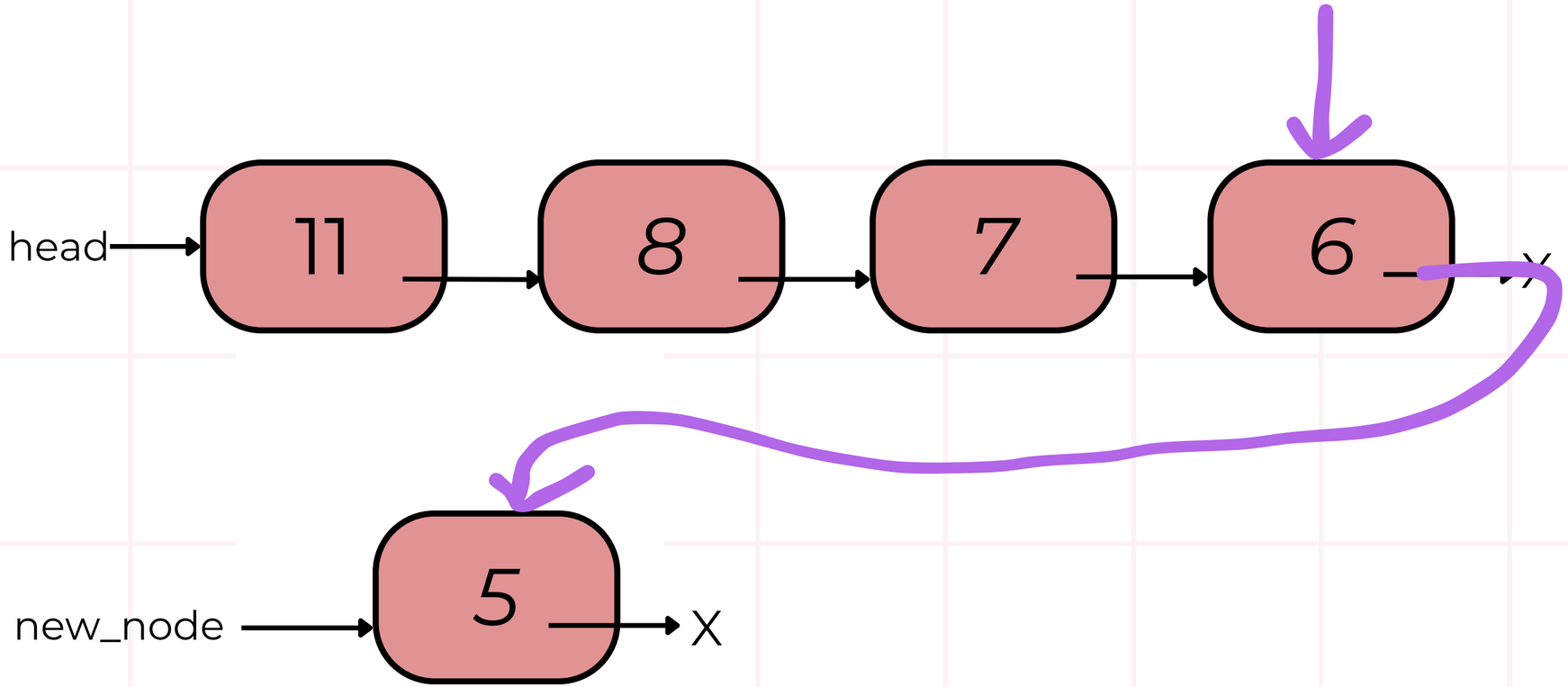
INSERT ANYWEHRE

EXAMPLE CASE #4: INSERT NEW NODE AT THE TAIL OF A NON-EMPTY LIST



INSERT ANYWEHRE

EXAMPLE CASE #4: INSERT NEW NODE AT THE TAIL OF A NON-EMPTY LIST



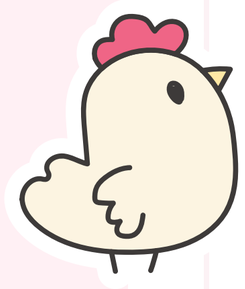
INSERT ANYWEHRE

EXAMPLE CASE #5: INVALID POSITION NUMBER (LESS THAN 0 OR GREATER THAN LENGTH)

Do nothing

NOW WE KNOW HOW TO INSERT...
HOW DO WE DELETE NODES?

GIVEN A LIST AND AN INTEGER VALUE, HOW DO WE DELETE THE VALUE FROM THE LIST IF IT EXISTS?
(ASSUMING VALUES IN THE LIST ARE UNIQUE)



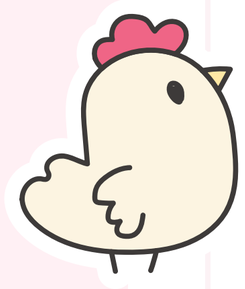
DELETE A SPECIFIC NODE

For example, given:

- the list,
  - 11->8->7->6->X
- an integer value of
  - 7

The list will become:

- 11->8->6->X



HOW DO WE REMOVE/DELETE A NODE?

Depends... like before, we need to consider some "cases":

- *How many nodes do we have in the list?*
  - *Empty list?*
    - *then there's nothing to delete!*
  - *Only one node in the list?*
  - *More than one / many nodes in the list?*

- *Which node in the list is to be deleted:*
  - *a node at the head? (i.e. first node)*
  - *a node between two other nodes?*
  - *a node at the tail? (i.e. last node)*
- ...

General steps to do this!

1. Find the previous node to the one that is being deleted
2. Make another pointer keep track of the node to delete (for step 4)
3. Change the next pointer of this previous node to skip the node to be deleted
4. Free the memory for the node we are deleting

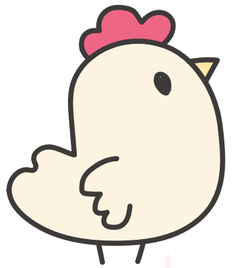
Don’t forget to also consider different edge cases

DIAGRAMS!

DIAGRAMS!

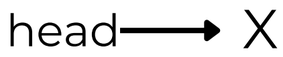
CODING TIME! (AGAIN)

Delete node



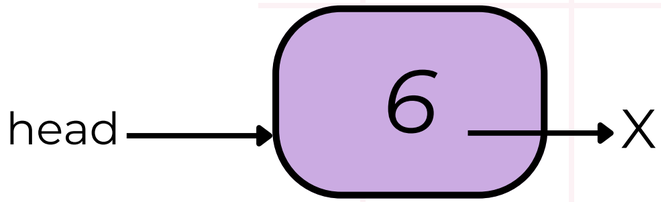
DELETE NODE

EXAMPLE CASE #1: DELETE SOME NODE IN AN EMPTY LIST



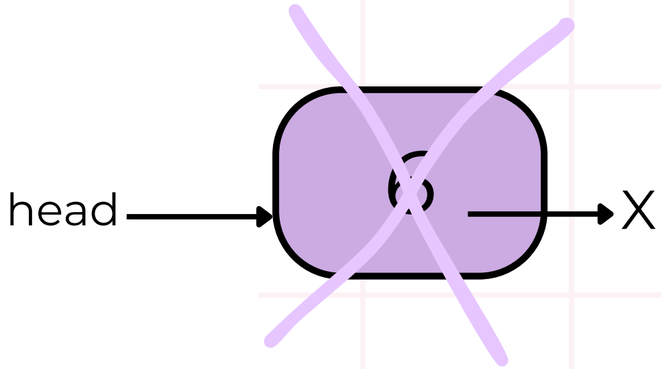
DELETE NODE

EXAMPLE CASE #2: DELETE THE NODE IN A LIST WITH ONLY ONE ELEMENT



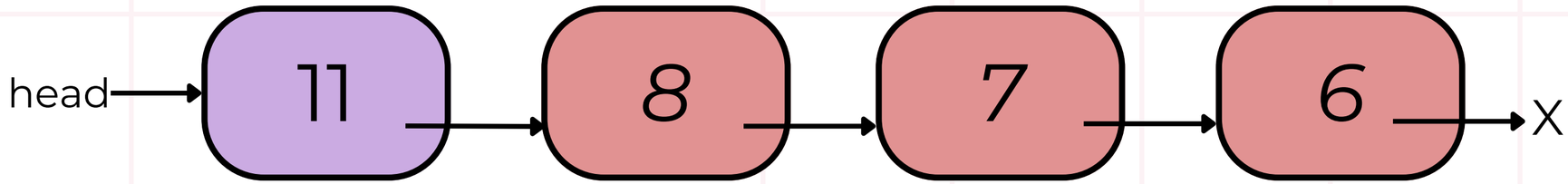
DELETE NODE

EXAMPLE CASE #2: DELETE THE NODE IN A LIST WITH ONLY ONE ELEMENT



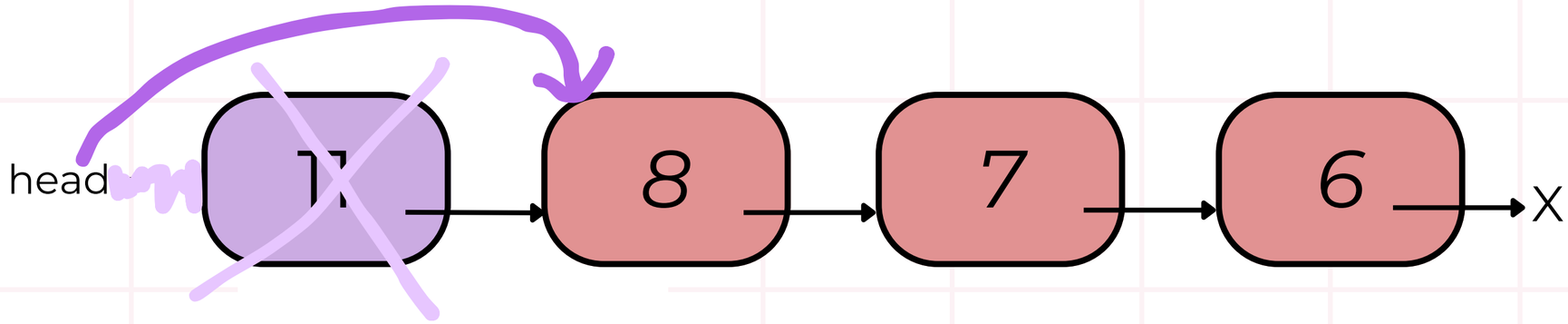
DELETE NODE

EXAMPLE CASE #3:
DELETE NODE
AT THE HEAD
IN A LIST WITH MULTIPLE NODES



DELETE NODE

EXAMPLE CASE #3:
DELETE NODE
AT THE HEAD
IN A LIST WITH MULTIPLE NODES



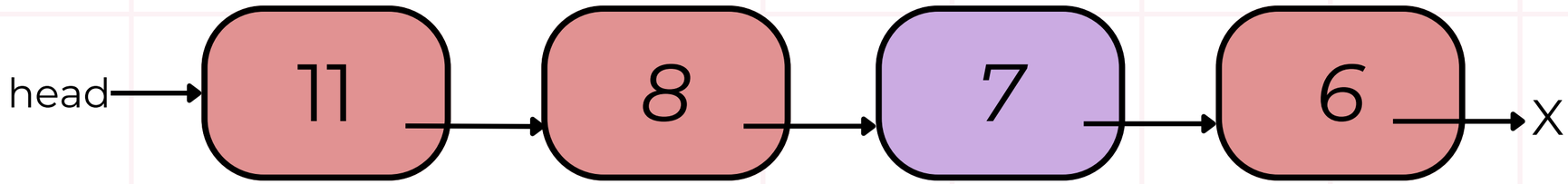
DELETE NODE

EXAMPLE CASE #4 (AVERAGE CASE):

DELETE SOME NODE

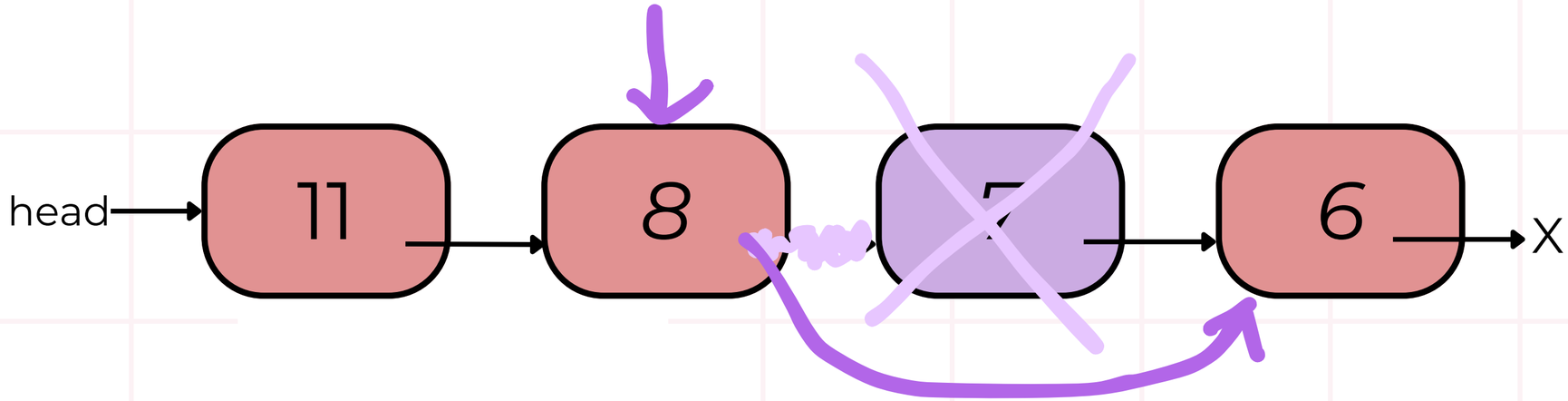
IN BETWEEN TWO NODES

IN A LIST WITH MULTIPLE NODES



DELETE NODE

EXAMPLE CASE #4 (AVERAGE CASE):
DELETE SOME NODE
IN BETWEEN TWO NODES
IN A LIST WITH MULTIPLE NODES



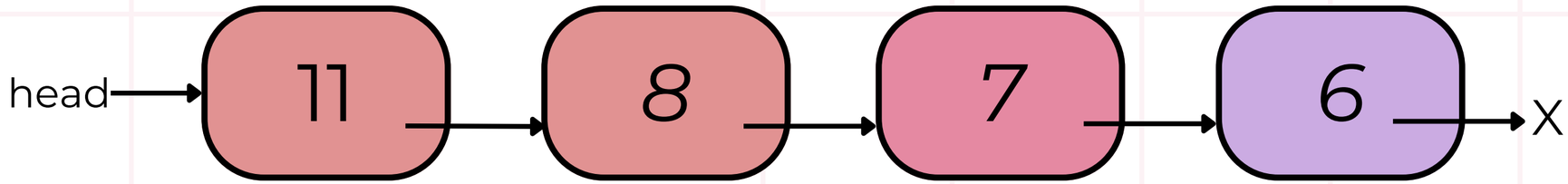
DELETE NODE

EXAMPLE CASE #5:

DELETE NODE

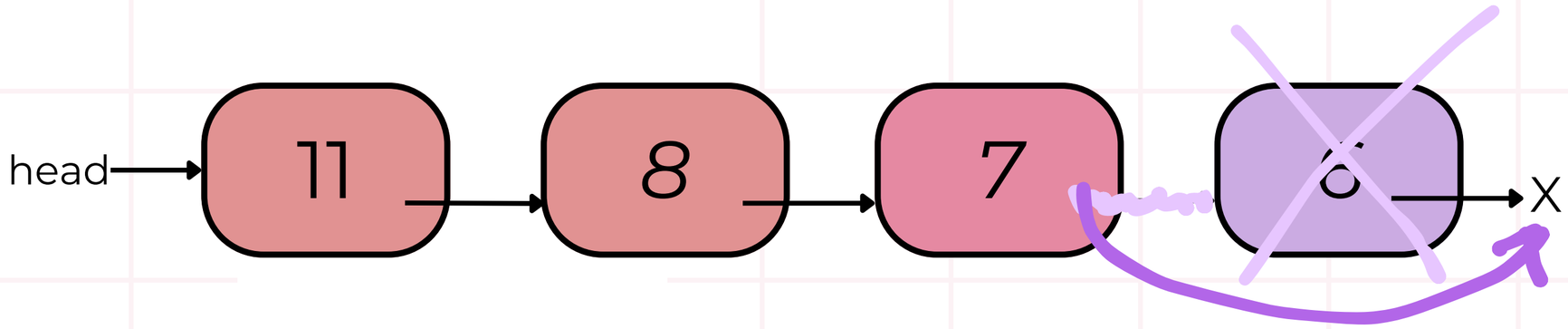
AT THE TAIL

IN A LIST WITH MULTIPLE NODES



DELETE NODE

EXAMPLE CASE #5:
DELETE NODE
AT THE TAIL
IN A LIST WITH MULTIPLE NODES



LINKED LIST CODE WRITING CHECKLIST :)

For any linked list operations you try and code up:

☑ *Are you drawing diagrams as you code -*
Draw, Code, Repeat! (It's so much easier to debug this way!)

☑ *Have you considered all the possible cases we can operate in? Here are <u>some</u> we mentioned that might apply:*

- *How many nodes do we have in the list?*
  - *Empty list?*
    - *then there's nothing to delete!*
  - *Only one node in the list?*
  - *More than one / many nodes in the list?*

- *Which node/where in the list do we want to operate on:*
  - *at the head?*
  - *between two nodes?*
  - *at the tail?*

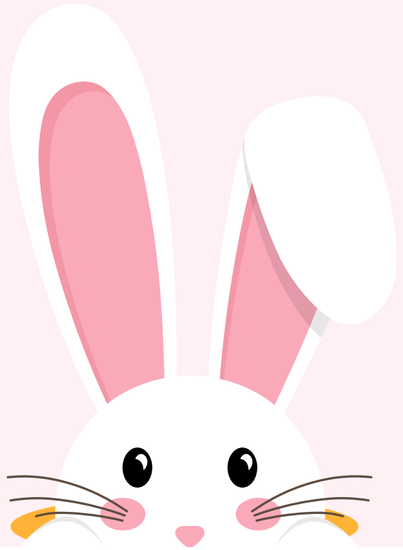
FAQ :)

When do we use malloc(...)?

- *when we have “new” data to be inserted into a list*
  - *working with existing data doesn’t count e.g. printing the list, we don’t need malloc(...)*

When do we use free(...)?

- *when we are trying to “remove” any node*
- *whenever we use malloc(...) in our programs, there should be a corresponding free(...) for each piece of memory malloc-ed*



FEEDBACK (PRETTY PLEASE WITH A CHERRY ON TOP)

Thank you for taking time to give me feedback

https://forms.office.com/r/Cn8FgdFPhu

Enjoy your Easter long weekend!
(No lecture next Monday)

LINKED LIST BUZZWORDS: NODE, NULL, HEAD, TAIL

SUMMARY OF TODAY

- More linked lists!
  - insert at head (continued from last lecture)
  - traverse a linked list
  - insert at tail
  - insert anywhere
  - intro to deleting nodes (maybe)

NEXT LECTURE

(I.E. NEXT WEDNESDAY)

Even more linked list operations!

(with a focus on deletion of nodes in the list)

If you have any questions:

COURSE RELATED

COURSE FORUM + HELP SESSIONS!

ADMIN RELATED

CS1511@UNSW.EDU.AU

thank you

And come say hi if you see me around on campus :D