Digging deeper into linked lists

Where we left off

- You can create a linked list yourself
- It might look something like this:

```
head->[1|next]->[2|next]->[7|next]->!null
```

Which variable should be used to store the linked list?

A) `tail`

B) `node`

C) `node1`

D) `head`

E) none of the above

D

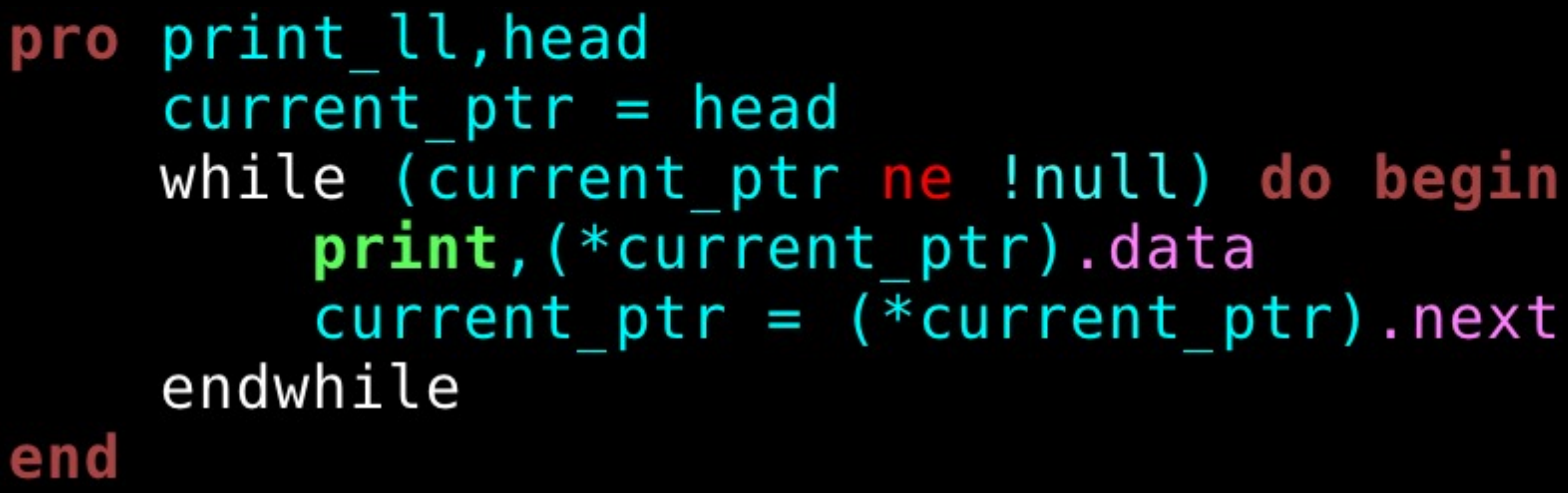
Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```



```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

```
print_ll,head
```

Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

`current_ptr = head`

Copies the *pointer* head. So now, we have:

```
current_ptr ->
 head->[1|next]->[2|next]->[7|next]->!null
```

Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

Since `current_ptr = head,` it is not `!null`

```
current_ptr ->
head->[1|next]->[2|next]->[7|next]->!null
```

Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

prints 1

```
current_ptr ->
head->[1|next]->[2|next]->[7|next]->!null
```

Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

Now `current_ptr` is re-assigned: it is a copy of the *pointer* in the first node

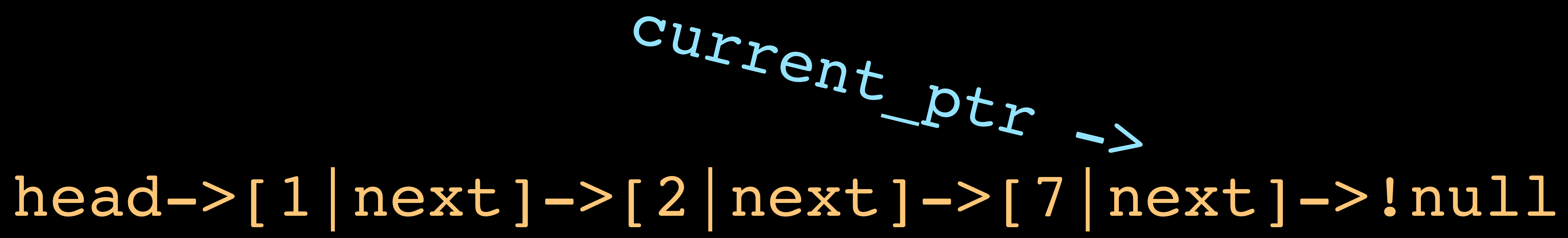
```
              current_ptr ->
head->[1|next]->[2|next]->[7|next]->!null
```

Traversing the list

```
head->[1|next]->[2|next]->[7|next]->!null
```

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

Repeat



Traversing the list

`head->[1|next]->[2|next]->[7|next]->!null`

```
pro print_ll,head
    current_ptr = head
    while (current_ptr ne !null) do begin
        print,(*current_ptr).data
        current_ptr = (*current_ptr).next
    endwhile
end
```

`current_ptr` is now `!null`, we’re done

`current_ptr ->`

`head->[1|next]->[2|next]->[7|next]->!null`

Will this work?

```
pro print_ll_bad,head
    current_ptr = *head
    while (current_ptr.next ne !null) do begin
        print,current_ptr.data
        current_ptr = *(current_ptr.next)
    endwhile
    print,current_ptr.data
end
```

A) Yes, it will do the same thing as the other `print_ll`

B) Yes, but it will do something different

C) No, it will crash

D) No, it is just bad code

E) None of the above

A. Yes, it will work, but only for printing (accessing) things, not for modifying them

Adding items

- You’ll write `add_head` and `add_tail` in tutorial
- We’ll go over the much more complicated `insert_ll` task now

Put 3 in this LL

```
head->[1|next]->[2|next]->[7|next]->!null
```

- To start, all we have defined is the `head` pointer

Where does our new node, `[3 | ptr]`, go?

```
head->[1|next]->[2|next]->[7|next]->!null
       A               B              C              D
```

E) None of the above

C

```
head->[1|next]->[2|next]->[7|next]->!null
```

How should we start our `insert_ll` procedure?

```
pro insert_ll,number,head
```

A)
```
prev = *head
next = prev.next
```

B)
```
prev = head
next = (*prev).next
```

C)
```
current = *head
next = *(current.next)
```

D)
```
current = *head
next = *(current).next
```

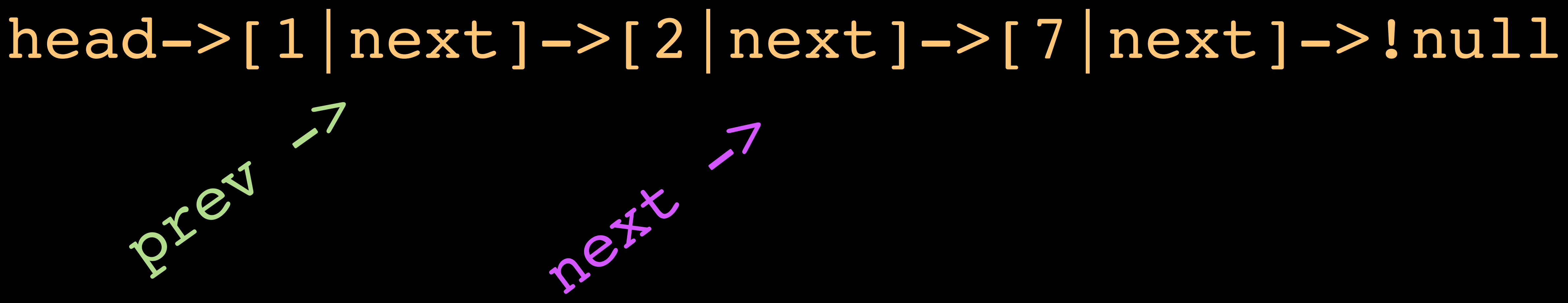
E) None of the above

B

Pointers for Inserting

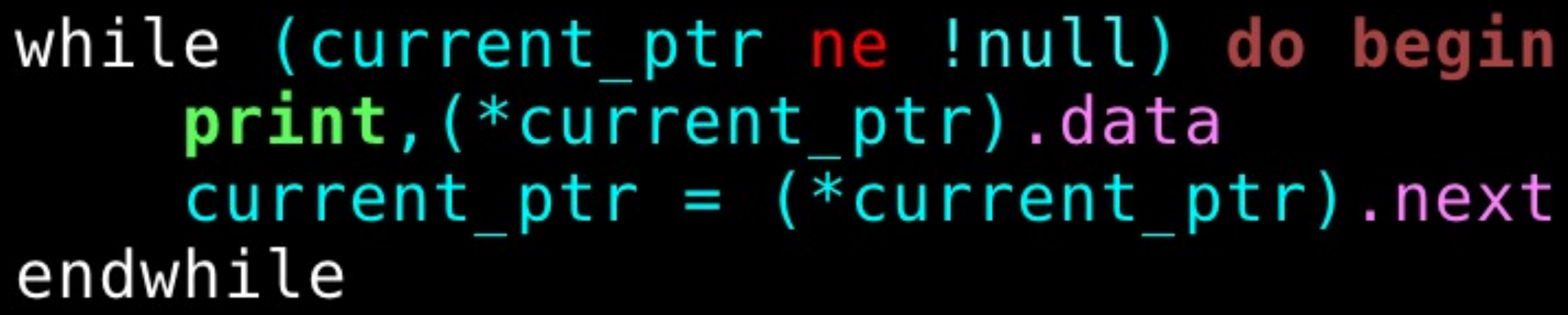
```
head->[1|next]->[2|next]->[7|next]->!null
```

```
prev = head
next = (*prev).next
```



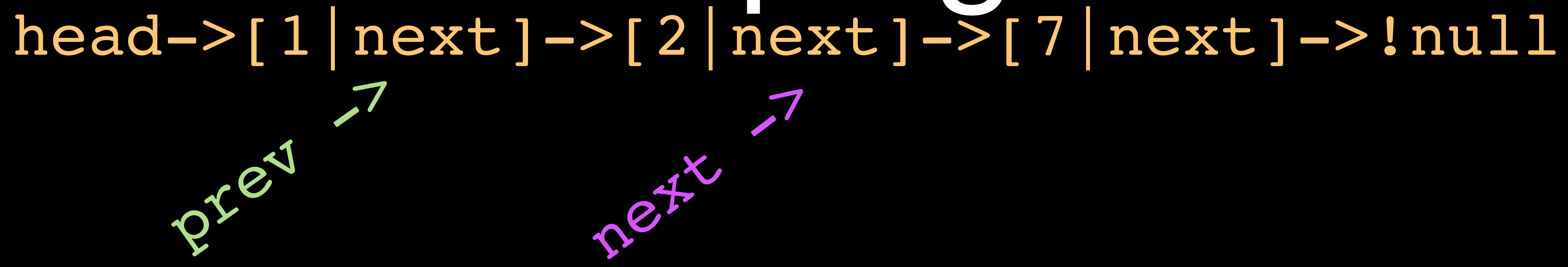
What about our loop?

- For now, let’s start with the same kind of loop we used for `print_ll`: loop until the end of the LL is reached



```
while (current_ptr ne !null) do begin
    print,(*current_ptr).data
    current_ptr = (*current_ptr).next
endwhile
```

Looping

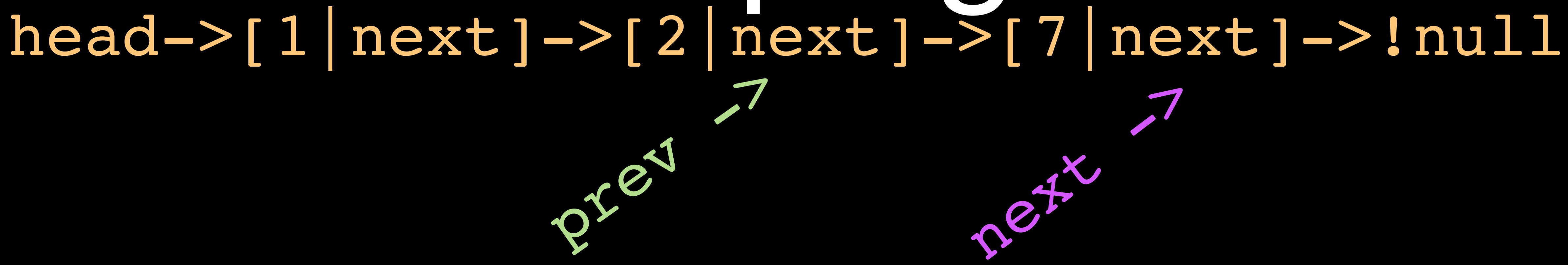


```
head->[1|next]->[2|next]->[7|next]->!null
```

```
; loop through until non-null
while (next ne !null) do begin


    prev = next
    next = (*next).next
endwhile
```

Looping



```
; loop through until non-null
while (next ne !null) do begin


    prev = next
    next = (*next).next
endwhile
```

Looping

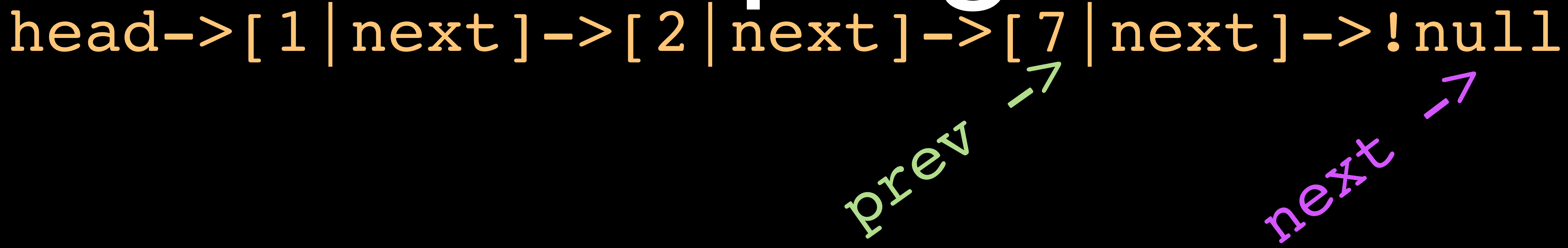
```
head->[1|next]->[2|next]->[7|next]->!null
                      prev ->       next ->
```

```
; loop through until non-null
while (next ne !null) do begin

    prev = next
    next = (*next).next
endwhile
```

DONE!

Looping



```
; loop through until non-null
while (next ne !null) do begin
```

Clearly, we need to do something in here though
Something involving data comparison...

```
    prev = next
    next = (*next).next
endwhile
```

```
head->[1|next]->[2|next]->[7|next]->!null
```

prev ->

next ->

Which data should we compare against?

A) `(*prev).data`

B) `(*next).data`

C) Either A or B

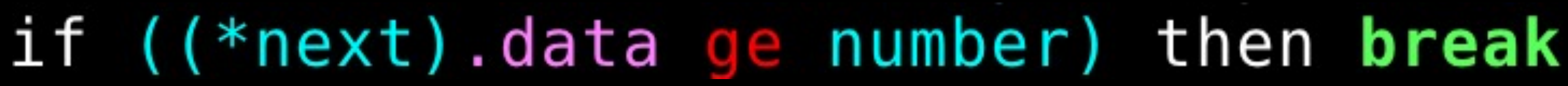
D) `(*head).data`

E) All of the above

C works, but A is a bonus because of special cases

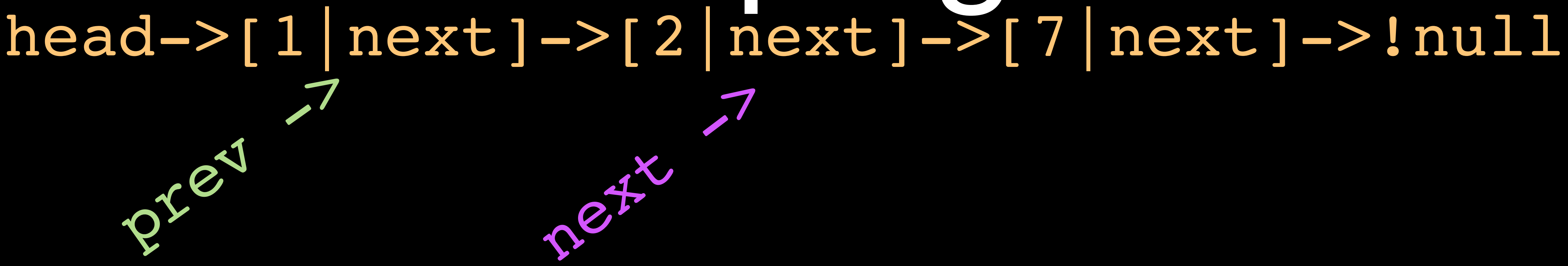
Breaking out of the loop

- We want to take our action - inserting the new data into the LL - only once
- Therefore, it shouldn’t be part of the loop. But, we want the loop to quit when we meet the right condition



```
if ((*next).data ge number) then break
```

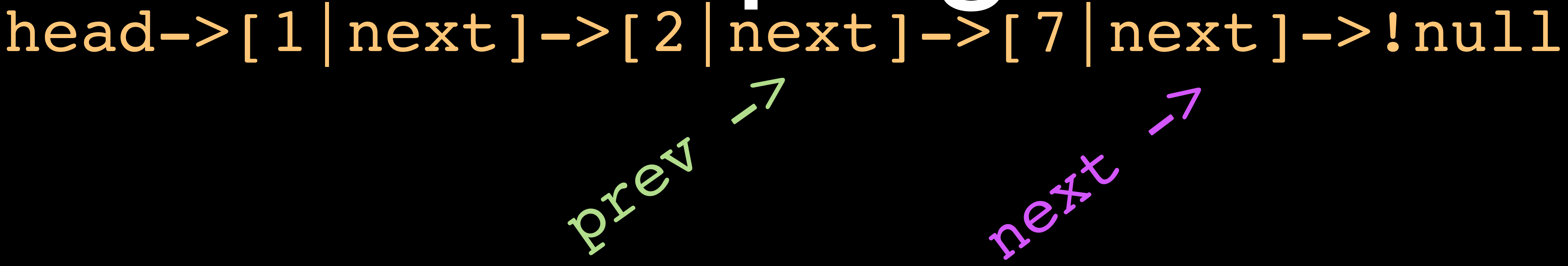
Looping



```
; loop through until non-null
while (next ne !null) do begin
    ; really, this is part of the "while" condition
    ; but IDL will crash if you try to deref a !null
    if ((*next).data ge number) then break
    prev = next
    next = (*next).next
endwhile
```

Looping again...

Looping



```
; loop through until non-null
while (next ne !null) do begin
    ; really, this is part of the "while" condition
    ; but IDL will crash if you try to deref a !null
→   if ((*next).data ge number) then break
    prev = next
    next = (*next).next
endwhile
```

We’ve met the condition

Now modifying the LL

```
head->[1|next]->[2|next]->[7|next]->!null
              prev ->          next ->
```

- `[2|next]` -> (something new)
- (something new) -> `[7|next]`

Now modifying the LL

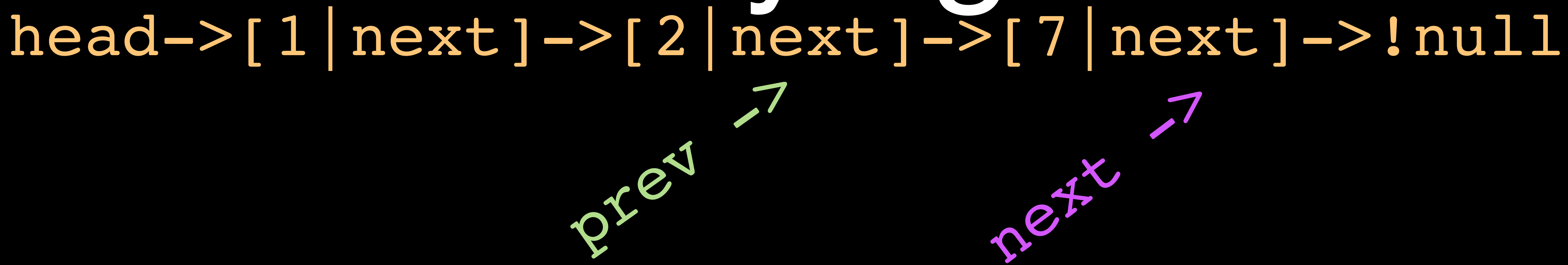
```
head->[1|next]->[2|next]->[7|next]->!null
```

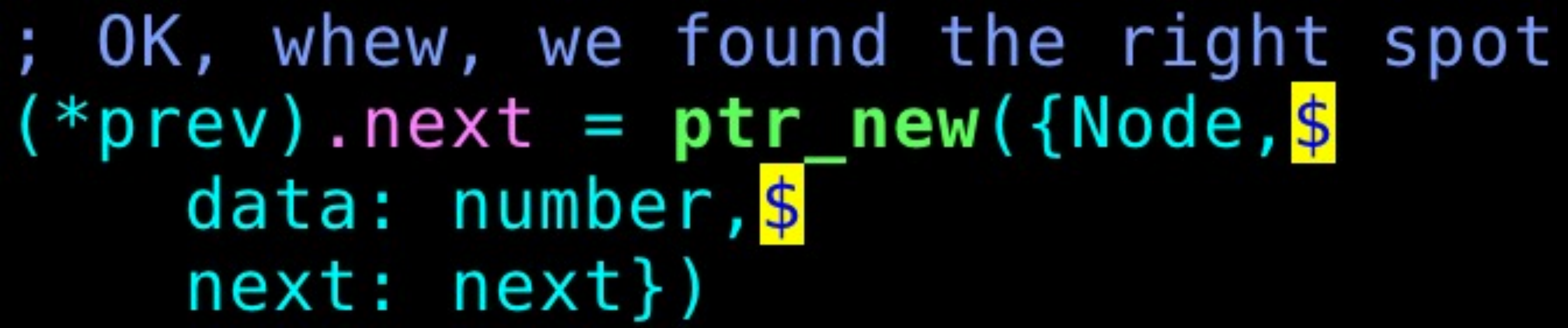
prev ->

next ->

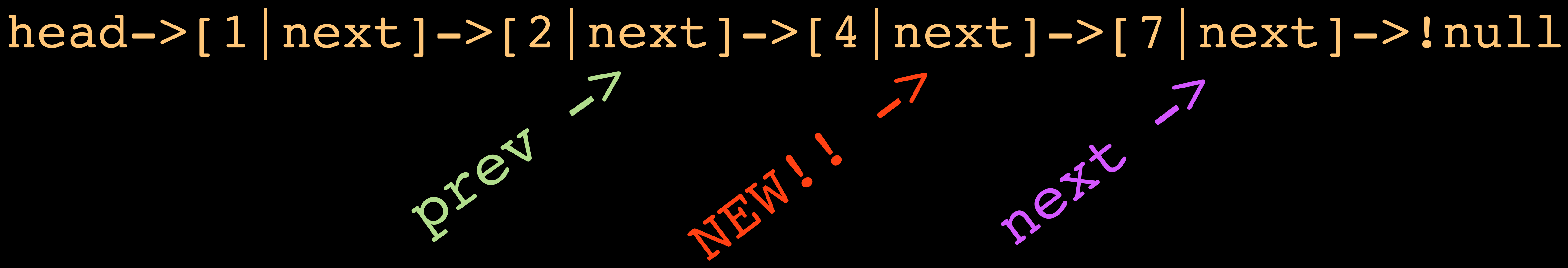
```
; OK, whew, we found the right spot
(*prev).next = ptr_new({Node,$
    data: number,$
    next: next})
```

Now modifying the LL





```
; OK, whew, we found the right spot
(*prev).next = ptr_new({Node,$
    data: number,$
    next: next})
```



Tricky cases

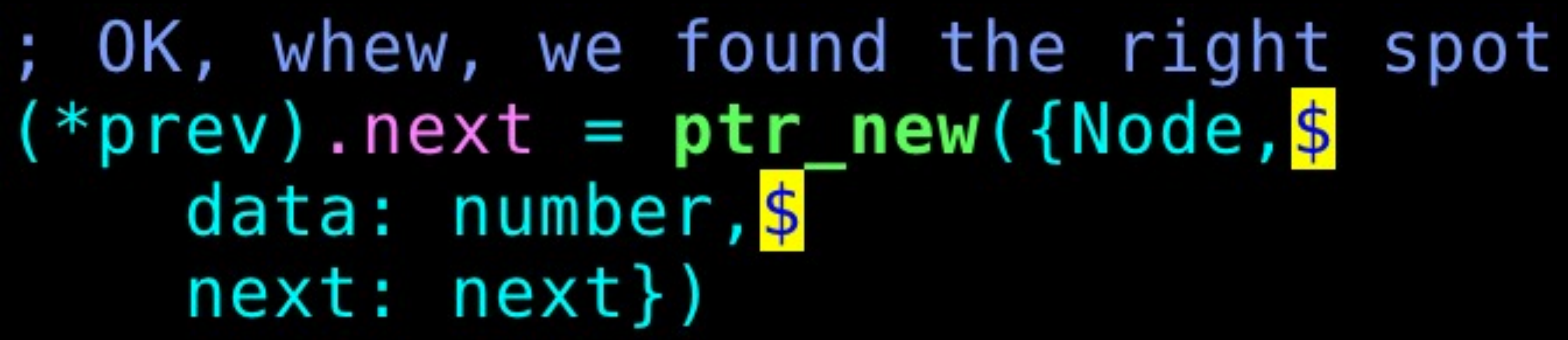
- What about when you want to add a number, and it’s greater than or less than all elements in the LL?
    - One of these cases is pretty easy: if it’s at the end, you’ll never meet the “break” condition, so you don’t have to do anything special

Adding 8 at the end

```
head->[1|next]->[2|next]->[7|next]->!null
```

prev ->

next ->



```
; OK, whew, we found the right spot
(*prev).next = ptr_new({Node,$
    data: number,$
    next: next})
```

```
head->[1|next]->[2|next]->[7|next]->[8|next]->!null
```

prev ->

NEW!! ->

next ->

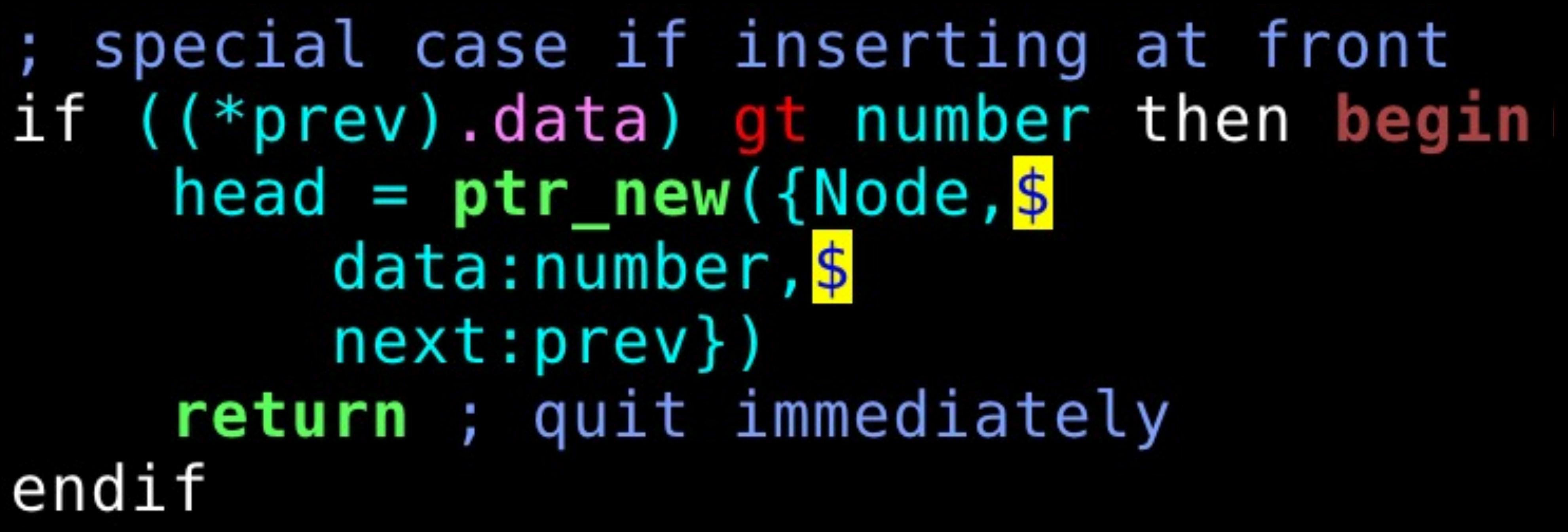
What about the beginning?

- You need an `if` statement.
  - You never need to set `next`, you just need to change `head`

```
; special case if inserting at front
if ((*prev).data) gt number then begin
    head = ptr_new({Node})
    (*head).next = prev
    (*head).data = number
    return ; quit immediately
endif
```

What about the beginning?

- You need an `if` statement.
    - You never need to set `next`, you just need to change `head`



```
; special case if inserting at front
if ((*prev).data) gt number then begin
    head = ptr_new({Node,$
        data:number,$
        next:prev})
    return ; quit immediately
endif
```

(equivalent)

Tutorial 19 has been updated

- Grab the latest from the website
  - only major change is that `insert_node` has been added
  - This lecture described *exactly* what should go in to `insert_node`, but it’s left as an exercise for you to write it up
  - Do `add_head`, `add_tail`, and `n_elements_ll` first, though - they’re easier