COMP108
Data Structures and Algorithms

Data structures - Linked Lists (Part III Deletion)

Professor Prudence Wong

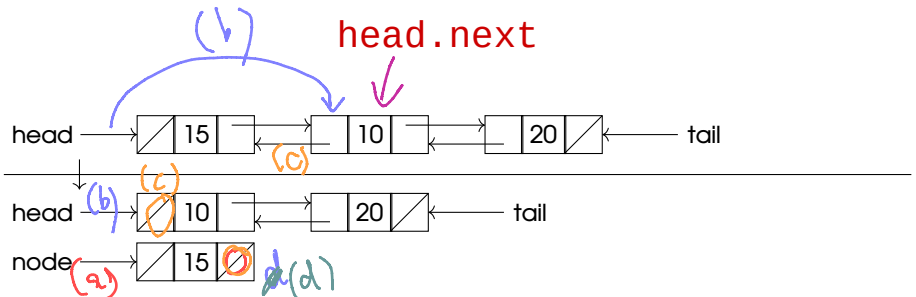
pwong@liverpool.ac.uk

2022-23

Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

List-Delete-Head(L)



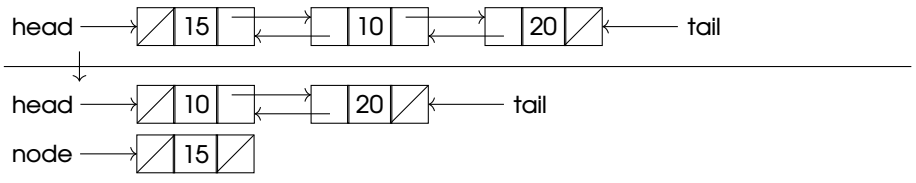
Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

List-Delete-Head(L)

node $\leftarrow$ head

return node



Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

List-Delete-Head(L)

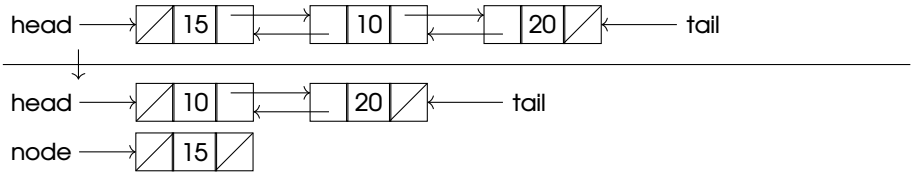
node $\leftarrow$ head

if node $\neq$ NIL then

begin // list wasn't empty

end

return node



Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

List-Delete-Head(L)

node $\leftarrow$ head

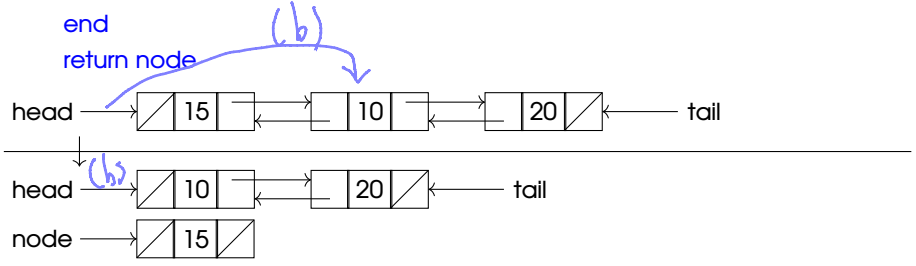
if node $\neq$ NIL then

begin // list wasn't empty

(b) head $\leftarrow$ head.next

end

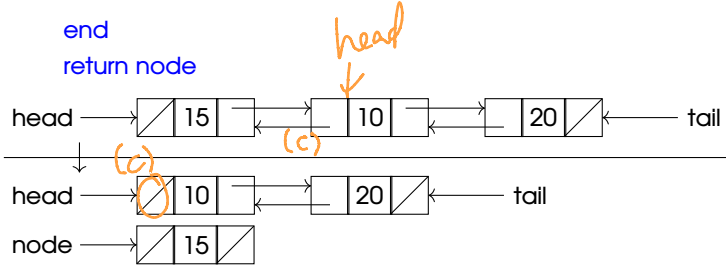
return node

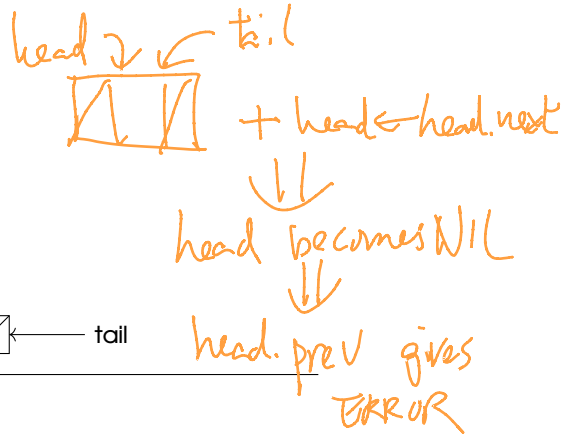


Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

```
List-Delete-Head(L)
    node ← head
    if node ≠ NIL then
    begin // list wasn't empty
        head ← head.next
        if head ≠ NIL then
            head.prev ← NIL
    end
    return node
```

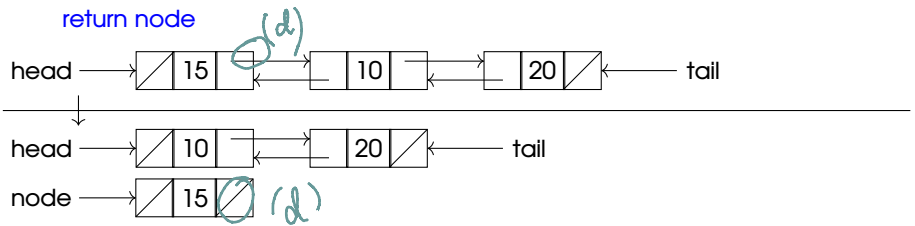




Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

```
List-Delete-Head(L)
    node ← head
    if node ≠ NIL then
    begin // list wasn't empty
        head ← head.next
        if head ≠ NIL then
            head.prev ← NIL
        node.next ← NIL
    end
    return node
```

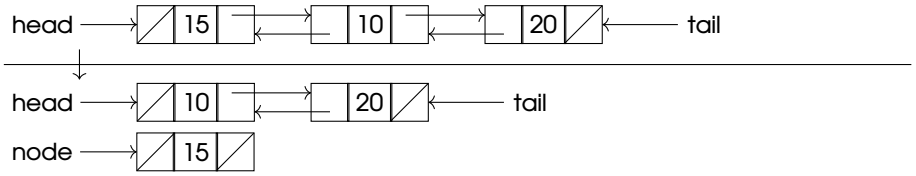


Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

```
List-Delete-Head(L)
    node ← head
    if node ≠ NIL then
    begin // list wasn't empty
        head ← head.next
        if head ≠ NIL then
            head.prev ← NIL
        node.next ← NIL
    end
    return node
```

- Order of statements is extremely important

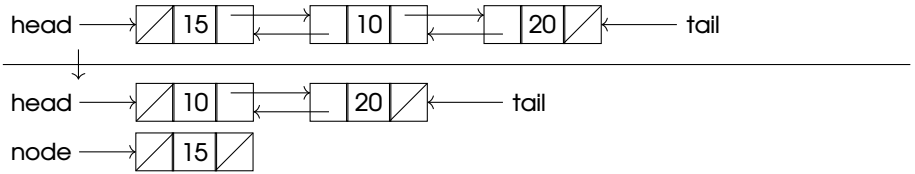


Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

```
List-Delete-Head(L)
    node ← head
    if node ≠ NIL then
    begin // list wasn't empty
        head ← head.next
        if head ≠ NIL then
            head.prev ← NIL
        node.next ← NIL
    end
    return node
```

- Order of statements is extremely important
- Updating `head` before `node`
  $\Longrightarrow$ `node` will point to 10 instead of 15.

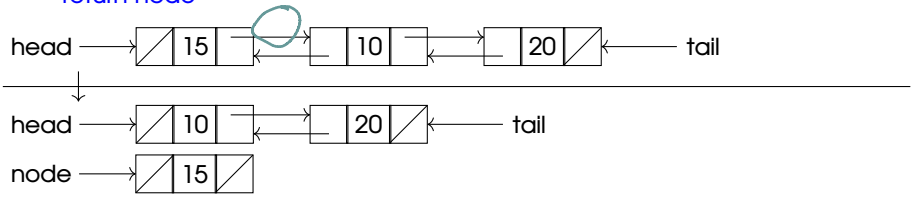


Linked lists - Algorithm - Deletion of a node from the front of the list

We want: a) return 15; b) head $\Longrightarrow$ 10; c) prev of 10 $\Longrightarrow$ NIL; d) next of 15 $\Longrightarrow$ NIL

```
List-Delete-Head(L)
1   node ← head
2   if node ≠ NIL then
3   begin // list wasn't empty
4       head ← head.next
5       if head ≠ NIL then
6           head.prev ← NIL
7       node.next ← NIL
8   end
9   return node
```

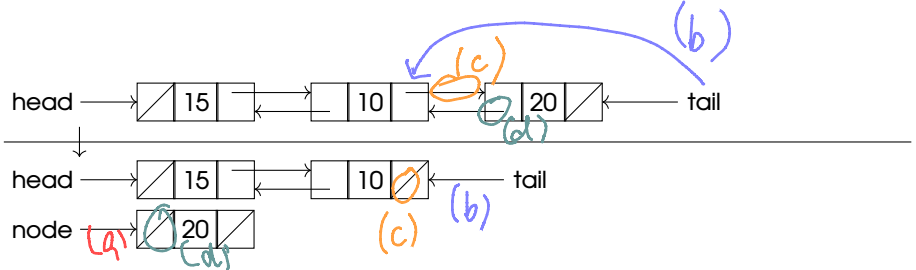
- Order of statements is extremely important
- Updating `head` before `node`
  $\Longrightarrow$ `node` will point to 10 instead of 15.
- Moving `node.next ← NIL` to line 2
  $\Longrightarrow$ the linked list is basically lost from the head



Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

List-Delete-Tail(L)



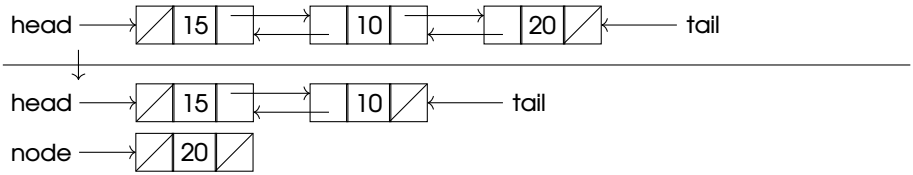
Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

List-Delete-Tail(L)

node $\leftarrow$ tail

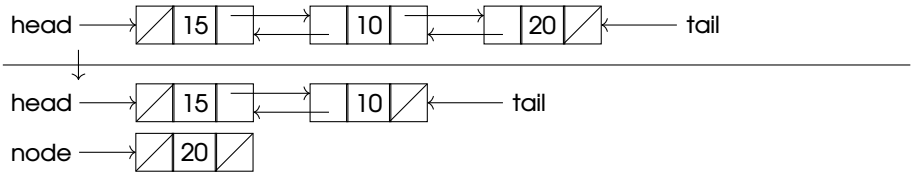
return node



Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

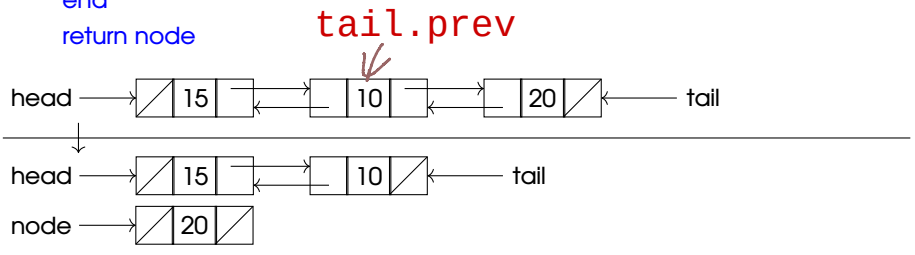
```
List-Delete-Tail(L)
    node ← tail
    if tail ≠ NIL then
    begin // list wasn't empty



    end
    return node
```



Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

```
List-Delete-Tail(L)
    node ← tail
    if tail ≠ NIL then
    begin // list wasn't empty
        tail ← tail.prev
        tail.next ← NIL
        node.prev ← NIL
    end
    return node
```

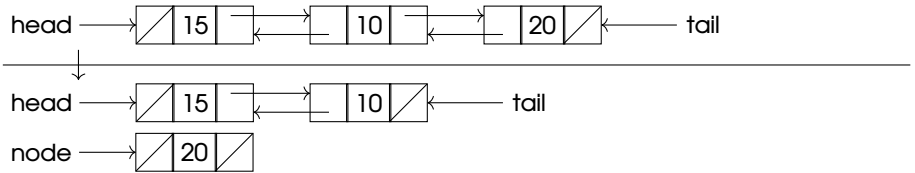


Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

```
List-Delete-Tail(L)
    node ← tail
    if tail ≠ NIL then
    begin // list wasn't empty
        tail ← tail.prev
        tail.next ← NIL
        node.prev ← NIL
    end
    return node
```

- Again, order of statements is extremely important

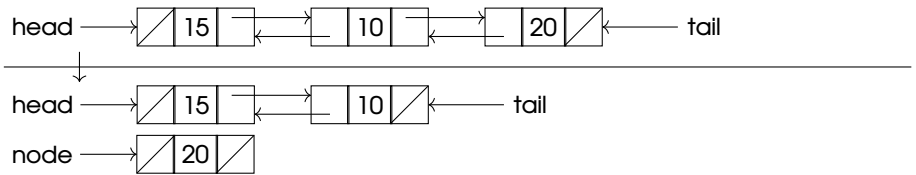


Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

```
List-Delete-Tail(L)
    node ← tail
    if tail ≠ NIL then
    begin // list wasn't empty
        tail ← tail.prev
        tail.next ← NIL
        node.prev ← NIL
    end
    return node
```

- Again, order of statements is extremely important
- Updating `tail` before `node`
  $\Longrightarrow$ `node` will point to 10 instead of 20.

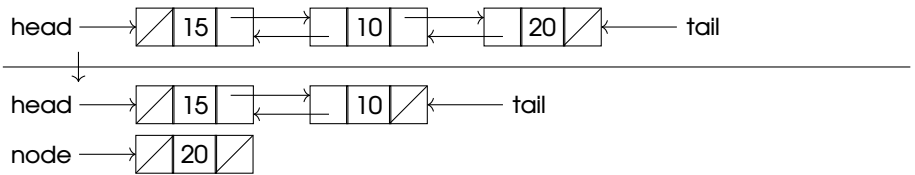


Linked lists - Algorithm - Deletion of a node from the tail of the list

We want: a) return 20; b) tail $\Longrightarrow$ 10; c) next of 10 $\Longrightarrow$ NIL; d) prev of 20 $\Longrightarrow$ NIL

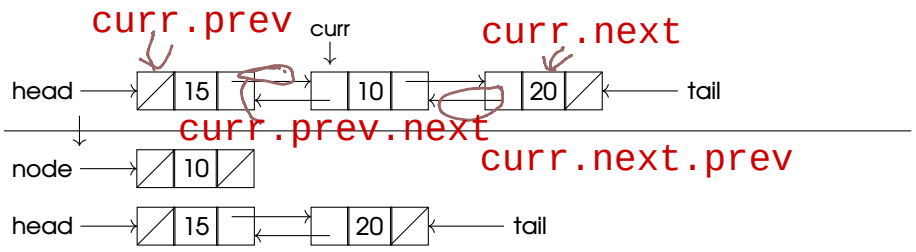
```
List-Delete-Tail(L)
    node ← tail
    if tail ≠ NIL then
    begin // list wasn't empty
        tail ← tail.prev
        tail.next ← NIL
        node.prev ← NIL
    end
    return node
```

- Again, order of statements is extremely important
- Updating `tail` before `node`
  $\Longrightarrow$ `node` will point to 10 instead of 20.
- Moving `node.prev ← NIL` to line 2
  $\Longrightarrow$ the linked list is basically lost from the tail



Linked list - Algorithm - Deletion from the middle

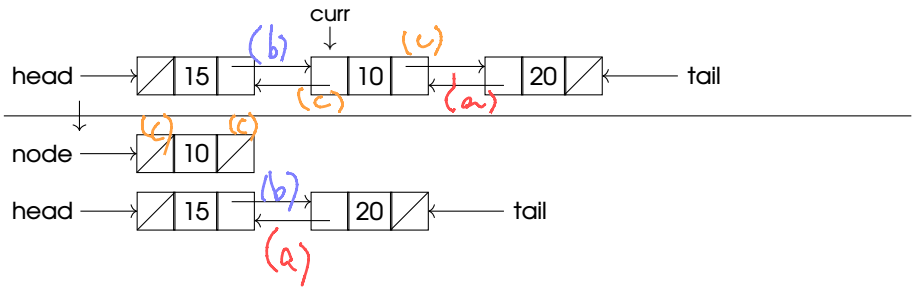
Suppose we want to delete a node somewhere in the list, say the node pointed to by **curr**



Linked list - Algorithm - Deletion from the middle

Suppose we want to delete a node somewhere in the list, say the node pointed to by **curr**

Want: a) prev of 20 $\Longrightarrow$ 15; b) next of 15 $\Longrightarrow$ 20; c) prev and next of 10 $\Longrightarrow$ NIL; d) return 10

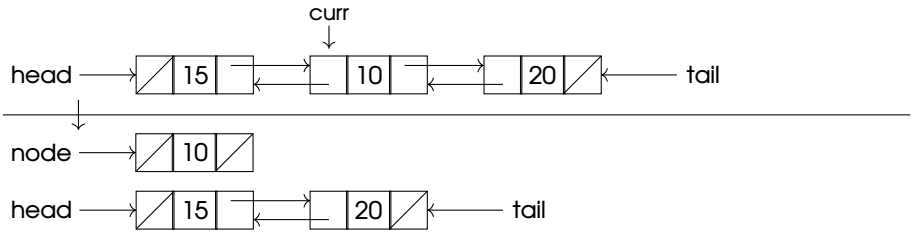



```
curr.next.prev <- curr.prev
curr.prev.next <- curr.next
curr.next <- NIL; curr.prev <- NIL
```

Linked list - Algorithm - Deletion from the middle

Suppose we want to delete a node somewhere in the list, say the node pointed to by **curr**

Want: a) prev of 20 $\Longrightarrow$ 15; b) next of 15 $\Longrightarrow$ 20; c) prev and next of 10 $\Longrightarrow$ NIL; d) return 10

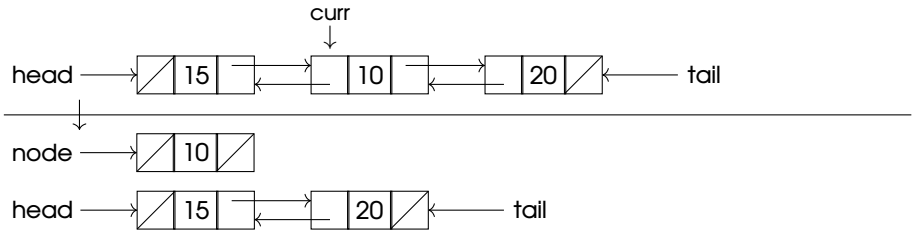



```
// assume that curr is actually pointing to a node
List-Delete(L, curr)
    curr.next.prev ⟵ curr.prev
    curr.prev.next ⟵ curr.next
```

Linked list - Algorithm - Deletion from the middle

Suppose we want to delete a node somewhere in the list, say the node pointed to by **curr**

Want: a) prev of 20 $\Longrightarrow$ 15; b) next of 15 $\Longrightarrow$ 20; c) prev and next of 10 $\Longrightarrow$ NIL; d) return 10

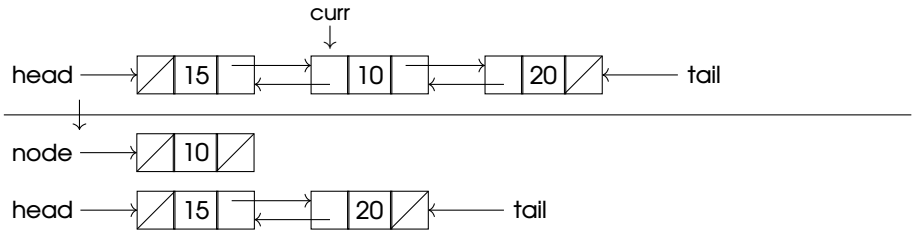



```
// assume that curr is actually pointing to a node
List-Delete(L, curr)
    curr.next.prev ← curr.prev
    curr.prev.next ← curr.next
    curr.next ← NIL
    curr.prev ← NIL
    return curr
```

Linked list - Algorithm - Deletion from the middle

Suppose we want to delete a node somewhere in the list, say the node pointed to by **curr**

Want: a) prev of 20 $\Longrightarrow$ 15; b) next of 15 $\Longrightarrow$ 20; c) prev and next of 10 $\Longrightarrow$ NIL; d) return 10




```
// assume that curr is actually pointing to a node
List-Delete(L, curr)
    curr.next.prev ← curr.prev
    curr.prev.next ← curr.next
    curr.next ← NIL
    curr.prev ← NIL
    return curr
```

What happen if `curr` is not in the middle?

Summary: Linked lists - Deletion

Next: Linked lists - Relationship with other data structures

For note taking