COMP108
Data Structures and Algorithms

Data structures - Linked Lists (Part II Insertion)

Professor Prudence Wong

pwong@liverpool.ac.uk

Reminder: Online Class Test on Thu 02 Mar

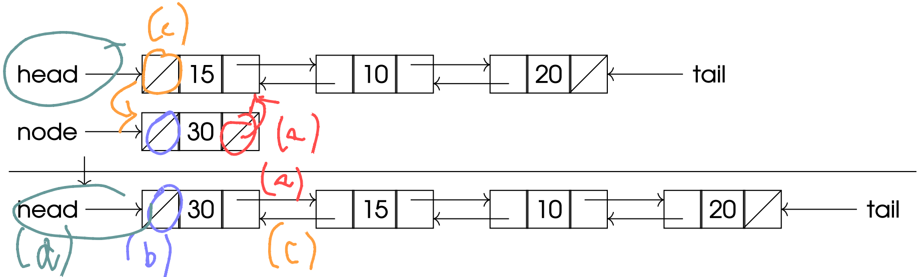
2022-23

revision lecture this Thu

Linked lists - Algorithm - Insertion of a node to the front of the list

Need: a) next of 30 $\Longrightarrow$ 15; b) prev of 30 $\Longrightarrow$ NIL; c) prev of 15 $\Longrightarrow$ 30; d) head $\Longrightarrow$ 30

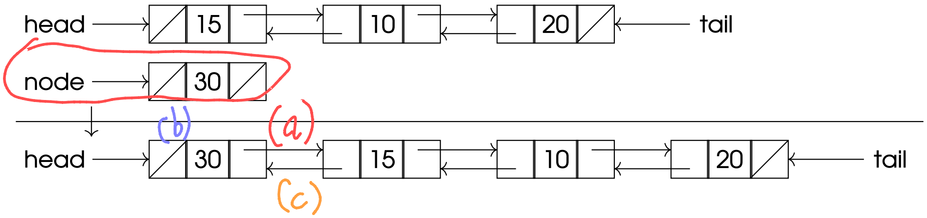
List-Insert-Head(L, node)



Linked lists - Algorithm - Insertion of a node to the front of the list

Need: a) next of 30 $\Longrightarrow$ 15; b) prev of 30 $\Longrightarrow$ NIL; c) prev of 15 $\Longrightarrow$ 30; d) head $\Longrightarrow$ 30

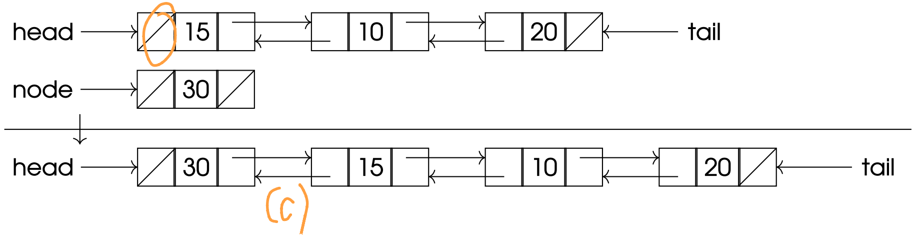
```
List-Insert-Head(L, node)
(a) node.next ← head
(b) node.prev ← NIL
```



Linked lists - Algorithm - Insertion of a node to the front of the list

Need: a) next of 30 $\Longrightarrow$ 15; b) prev of 30 $\Longrightarrow$ NIL; c) prev of 15 $\Longrightarrow$ 30; d) head $\Longrightarrow$ 30

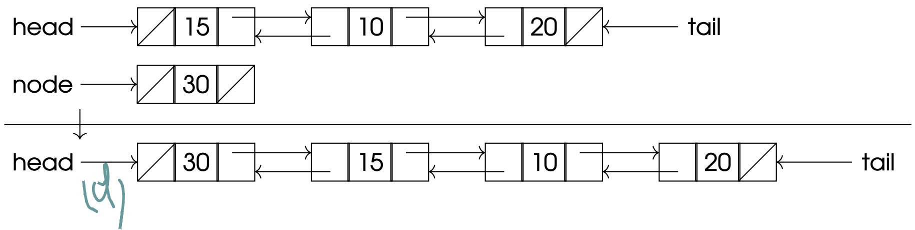
```
List-Insert-Head(L, node)
    node.next ← head
    node.prev ← NIL
    if head ≠ NIL then
(c)     head.prev ← node
    else // list was empty
        tail ← node
```



Linked lists - Algorithm - Insertion of a node to the front of the list

Need: a) next of 30 $\Longrightarrow$ 15; b) prev of 30 $\Longrightarrow$ NIL; c) prev of 15 $\Longrightarrow$ 30; d) head $\Longrightarrow$ 30

```
List-Insert-Head(L, node)
    node.next ← head
    node.prev ← NIL
    if head ≠ NIL then
        head.prev ← node
    else // list was empty
        tail ← node
(d) head ← node
```

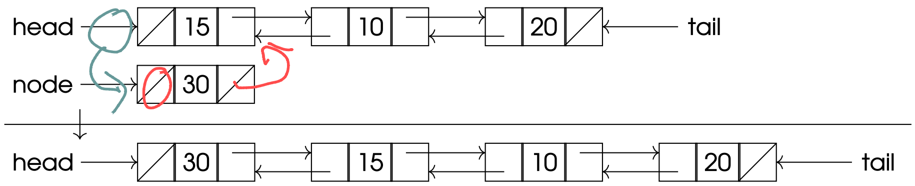


Linked lists - Algorithm - Insertion of a node to the front of the list

Need: a) next of 30 $\Longrightarrow$ 15; b) prev of 30 $\Longrightarrow$ NIL; c) prev of 15 $\Longrightarrow$ 30; d) head $\Longrightarrow$ 30

```
List-Insert-Head(L, node)
    node.next ← head
    node.prev ← NIL
    if head ≠ NIL then
        head.prev ← node
    else // list was empty
        tail ← node
    head ← node
```

Q. What happen if moving head ← node to line #3 before the if-statement?



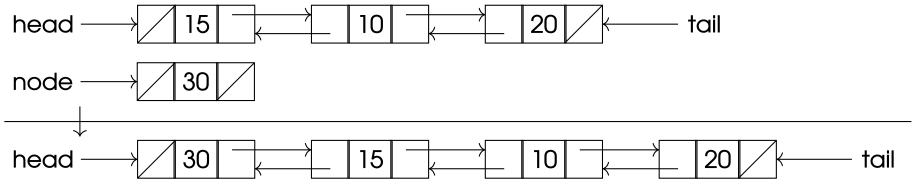
Linked lists - Algorithm - Insertion of a node to the front of the list

Need: a) next of 30 $\Longrightarrow$ 15; b) prev of 30 $\Longrightarrow$ NIL; c) prev of 15 $\Longrightarrow$ 30; d) head $\Longrightarrow$ 30

```
List-Insert-Head(L, node)
    node.next ← head
    node.prev ← NIL
    if head ≠ NIL then
        head.prev ← node
    else // list was empty
        tail ← node
    head ← node
```

Q. What happen if moving head ← node to line #3 before the if-statement?

A. We will lose pointer to the original list $\Longrightarrow$ broken list



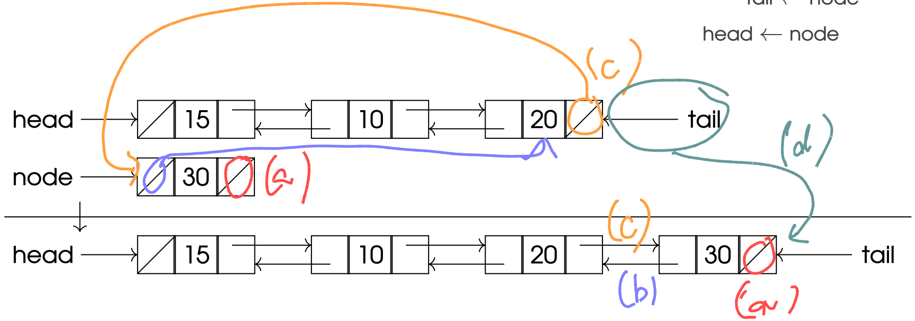
Linked lists - Algorithm - Insertion of a node to the tail of the list

Need: a) next of 30 $\Longrightarrow$ NIL; b) prev of 30 $\Longrightarrow$ 20; c) next of 20 $\Longrightarrow$ 30; d) tail $\Longrightarrow$ 30

List-Insert-Tail(L, node)

Comparing with List-Insert-Head(L, node)

```
node.next ← head
node.prev ← NIL
if head ≠ NIL then
    head.prev ← node
else // list was empty
    tail ← node
head ← node
```



Linked lists - Algorithm - Insertion of a node to the tail of the list

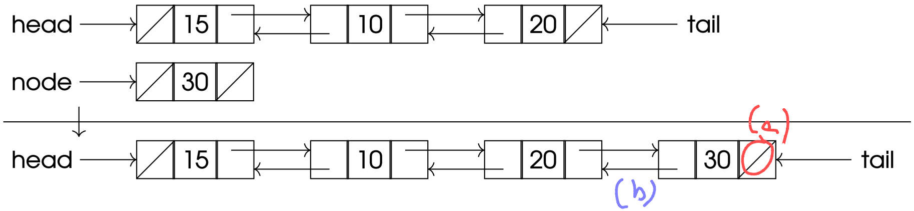
Need: a) next of 30 $\Longrightarrow$ NIL; b) prev of 30 $\Longrightarrow$ 20; c) next of 20 $\Longrightarrow$ 30; d) tail $\Longrightarrow$ 30

```
List-Insert-Tail(L, node)
(a) node.next ← NIL
(b) node.prev ← tail
```

Comparing with List-Insert-Head(L, node)

```
node.next ← head
node.prev ← NIL
if head ≠ NIL then
    head.prev ← node
else // list was empty
    tail ← node

head ← node
```



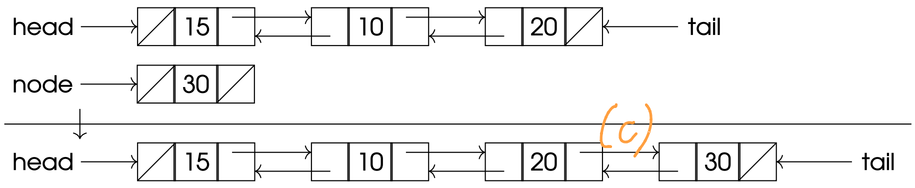
Linked lists - Algorithm - Insertion of a node to the tail of the list

Need: a) next of 30 $\Longrightarrow$ NIL; b) prev of 30 $\Longrightarrow$ 20; c) next of 20 $\Longrightarrow$ 30; d) tail $\Longrightarrow$ 30

```
List-Insert-Tail(L, node)
    node.next ← NIL
    node.prev ← tail
    if tail ≠ NIL then
(c)     tail.next ← node
    else // list was empty
        head ← node
```

Comparing with List-Insert-Head(L, node)

```
    node.next ← head
    node.prev ← NIL
    if head ≠ NIL then
        head.prev ← node
    else // list was empty
        tail ← node
    head ← node
```



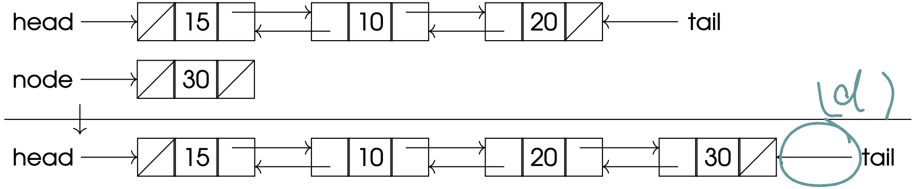
Linked lists - Algorithm - Insertion of a node to the tail of the list

Need: a) next of 30 $\Longrightarrow$ NIL; b) prev of 30 $\Longrightarrow$ 20; c) next of 20 $\Longrightarrow$ 30; d) tail $\Longrightarrow$ 30

```
List-Insert-Tail(L, node)
    node.next ← NIL
    node.prev ← tail
    if tail ≠ NIL then
        tail.next ← node
    else // list was empty
        head ← node
(d) tail ← node
```

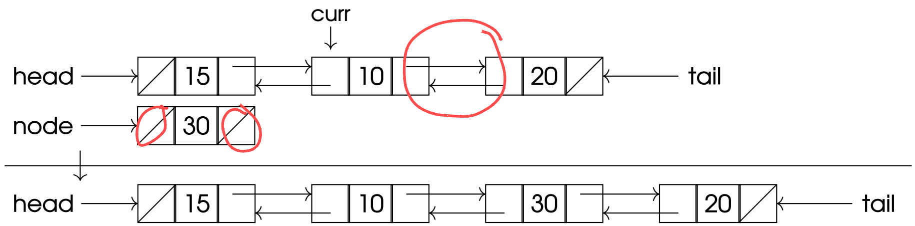
Comparing with List-Insert-Head(L, node)

```
node.next ← head
node.prev ← NIL
if head ≠ NIL then
    head.prev ← node
else // list was empty
    tail ← node
head ← node
```



Linked list - Algorithm - Insertion in the middle

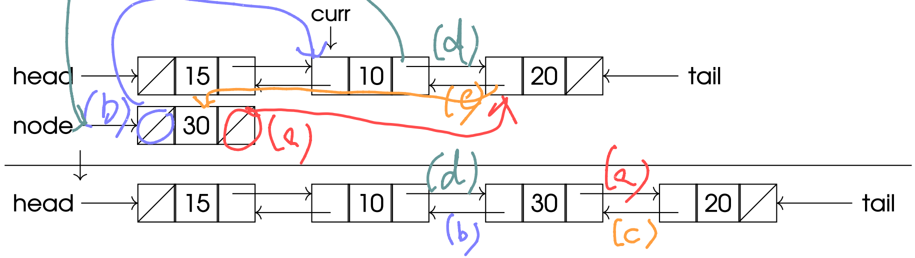
Suppose we want to insert a node somewhere in the list, say after a node pointed to by **curr**



Linked list - Algorithm - Insertion in the middle

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

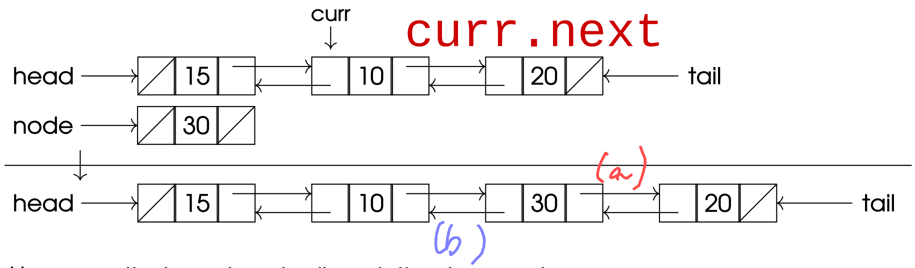
Need: a) next of 30 $\Longrightarrow$ 20; b) prev of 30 $\Longrightarrow$ 10; c) prev of 20 $\Longrightarrow$ 30; d) next of 10 $\Longrightarrow$ 30



Linked list - Algorithm - Insertion in the middle

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

Need: a) next of 30 $\Longrightarrow$ 20; b) prev of 30 $\Longrightarrow$ 10; c) prev of 20 $\Longrightarrow$ 30; d) next of 10 $\Longrightarrow$ 30

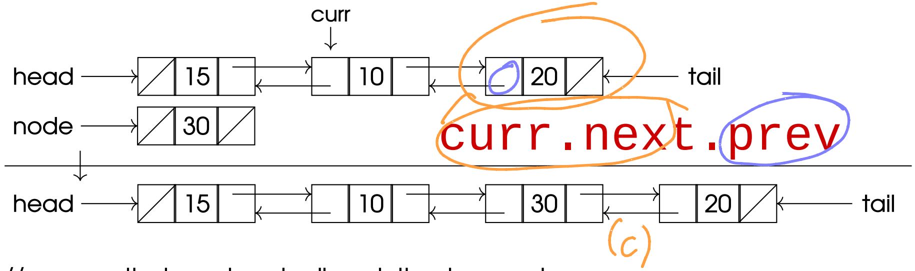


```
// assume that curr is actually pointing to a node
List-Insert(L, curr, node)
(a) node.next ← curr.next
(b) node.prev ← curr
```

Linked list - Algorithm - Insertion in the middle

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

Need: a) next of 30 $\Longrightarrow$ 20; b) prev of 30 $\Longrightarrow$ 10; c) prev of 20 $\Longrightarrow$ 30; d) next of 10 $\Longrightarrow$ 30

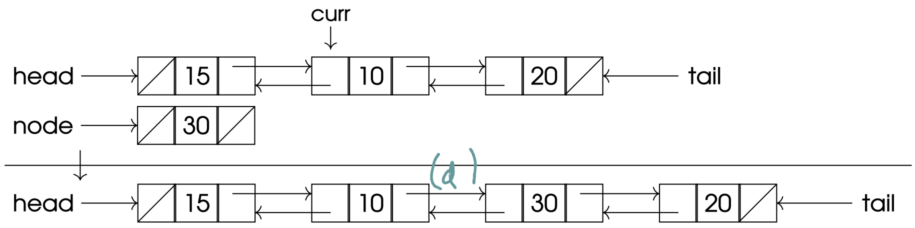


```
// assume that curr is actually pointing to a node
List-Insert(L, curr, node)
        node.next ← curr.next
        node.prev ← curr
(c)     curr.next.prev ← node
```

Linked list - Algorithm - Insertion in the middle

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

Need: a) next of 30 $\Longrightarrow$ 20; b) prev of 30 $\Longrightarrow$ 10; c) prev of 20 $\Longrightarrow$ 30; d) next of 10 $\Longrightarrow$ 30

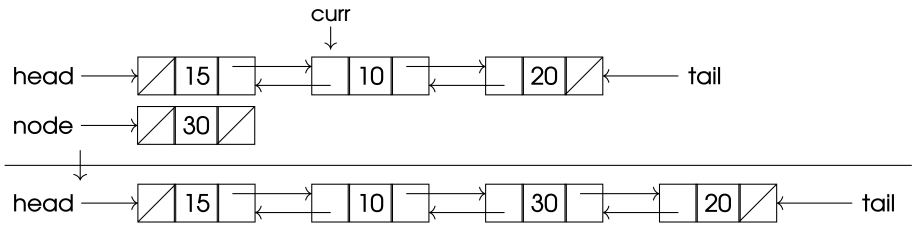


```
// assume that curr is actually pointing to a node
List-Insert(L, curr, node)
    node.next ← curr.next
    node.prev ← curr
    curr.next.prev ← node
    curr.next ← node
```

Linked list - Algorithm - Insertion in the middle

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

Need: a) next of 30 $\Longrightarrow$ 20; b) prev of 30 $\Longrightarrow$ 10; c) prev of 20 $\Longrightarrow$ 30; d) next of 10 $\Longrightarrow$ 30



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

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

Time complexity

Are the following statements correct about a linked list with *n* elements?

The time complexity of inserting an element to the head of a doubly linked list is $O(1)$. ✓

The time complexity of inserting an element to the tail of a doubly linked list is $O(1)$. ✓

The time complexity of inserting an element to the head of a singly linked list is $O(1)$. ✓

The time complexity of inserting an element to the tail of a singly linked list is $O(1)$. ✗

The time complexity of inserting an element to a sorted doubly linked list to maintain the order is $O(1)$.

```
finding position: O(n)
insert into that postion: O(1)
```

Summary: Linked lists - Insertion

Next: Linked lists - Deletion

For note taking