repl.it/@ckh205/SLLReverse

# Today's announcements:

HW1 due tonight ♡
PA1 out by noon tomorrow
due 2wk hence.

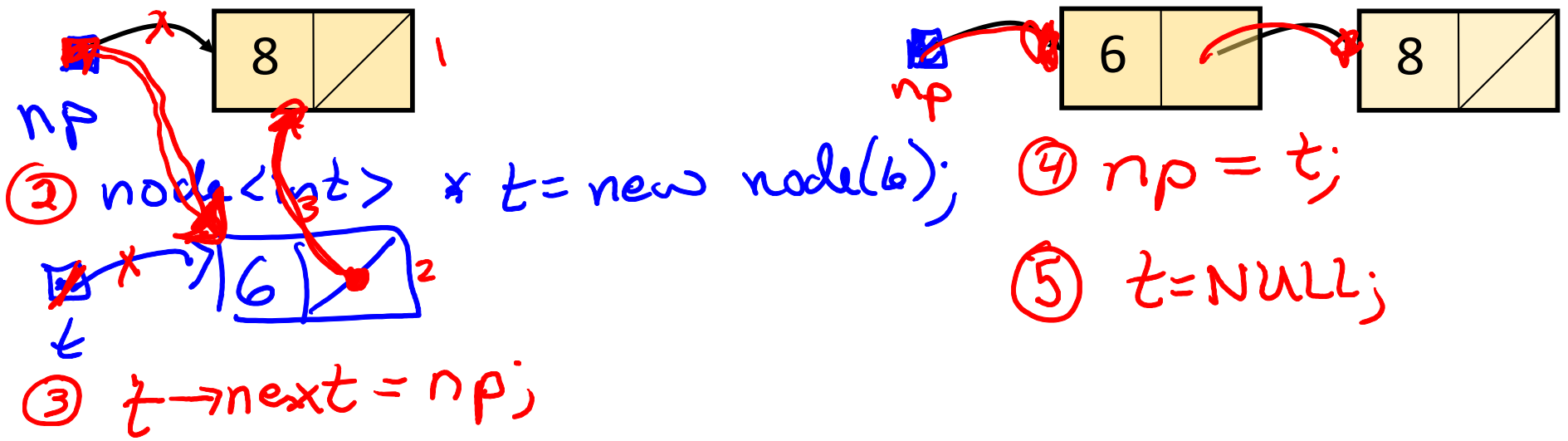
## A new memory model (continued):

```
1  class Node{
2     public:
3        LIT data;
4        Node * next;
5        Node(LIT newData):data(newData),next(NULL){} };
```

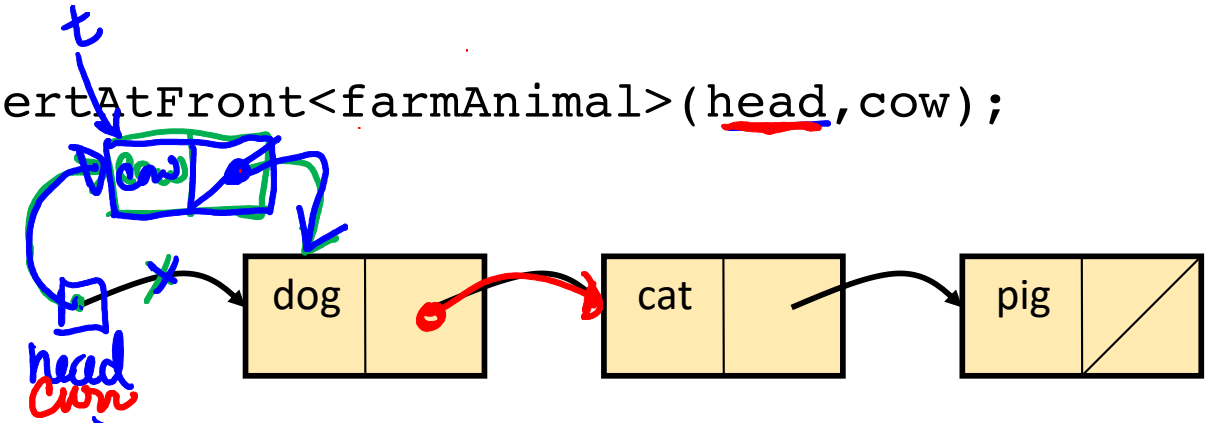
Node<int> nn(8);

① Node<int> * np = new node(8);

Write code that would result in each of these memory configurations (in order):



Example 1: `insertAtFront<farmAnimal>(head,cow);`




```
template<class LIT>
void insertAtFront(Node *& curr, LIT e) {
    node * t = new node(e);
    t->next= curr;
    Curr = t;
}
```

Running time?

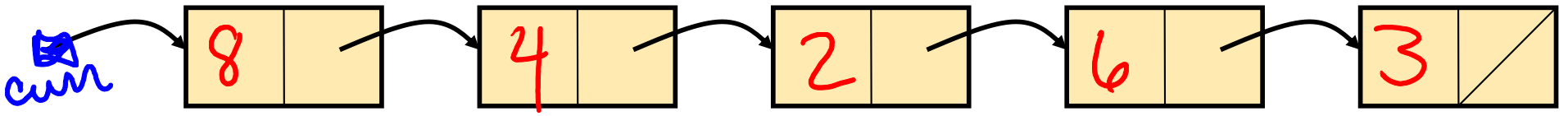
O(1)

```
1  struct Node{
2     LIT data;
3     Node * next;
4     Node(LIT newData):data(newData),next(NULL){}
5  };
```

Example 2:

8 4 2 6 3 — print reverse of rest of list, then 8.

36248




```
void printReverse(Node * curr) {
    if (curr != NULL) {
        printReverse(curr->next);
        cout << curr->data;
    }
}
```

aside:

$T(n) = T(n-1) + d$
$= T(n-2) + d + d$
$= T(n-k) + kd$
$= T(0) + nd$
$= (n+1)d$
$= O(n)$

nodes.

Running time?

$T(n)$: running time on a chain of $n$

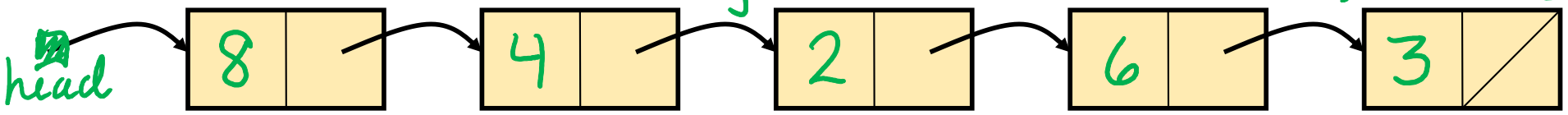
$n = 0$: $T(n) = d$

$n > 0$: $T(n) = T(n-1) + d$

8 4 2 6 3 → 3 2 8

Example 3:

You do & post to piazza.
My annotations, added 1/15, are in green.

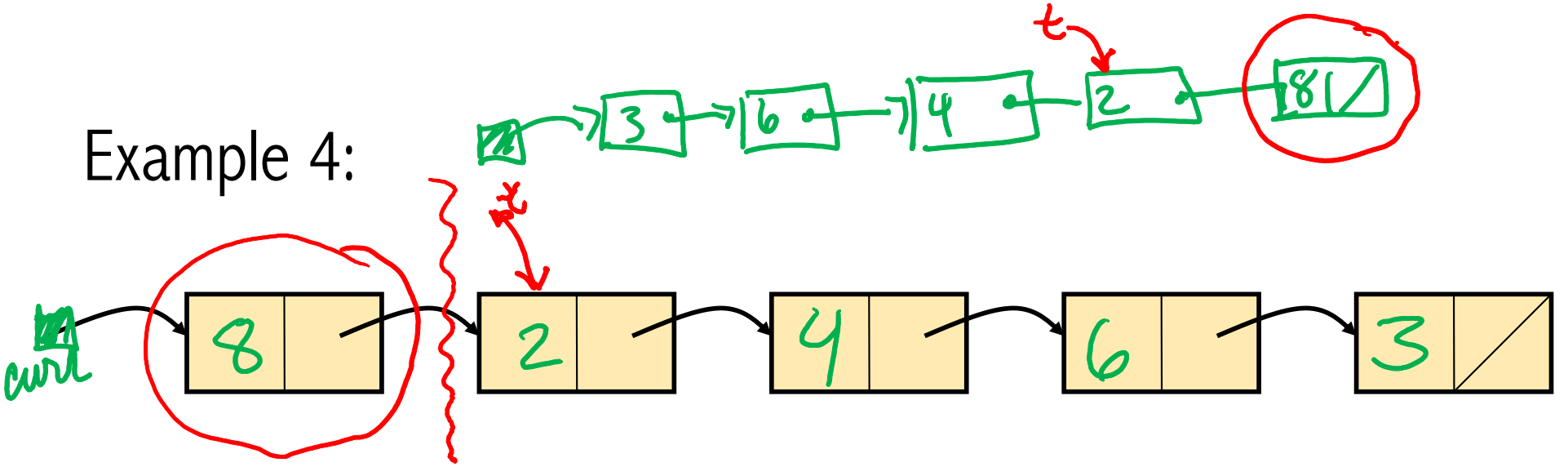



```
void printReverseODDS(Node * curr) {
    if (curr != NULL) {
        if (curr->next != NULL) {
            printReverseODDS(curr->next->next);
        }
        cout << curr->data << endl;
    }
}
```

Running time? O(n)

Recurrence:
$T(n) = T(n-2) + d$
$T(1) = d$, $T(0) = d$,

Example 4:




```
node * reverse(Node * curr) {
    if (curr != NULL && curr->next != NULL) {
        node * t = curr->next;
        node * rR = reverse(curr->next);
        t->next = curr;
        curr->next = NULL;
        curr = rR;
    }
    return curr;
}
```

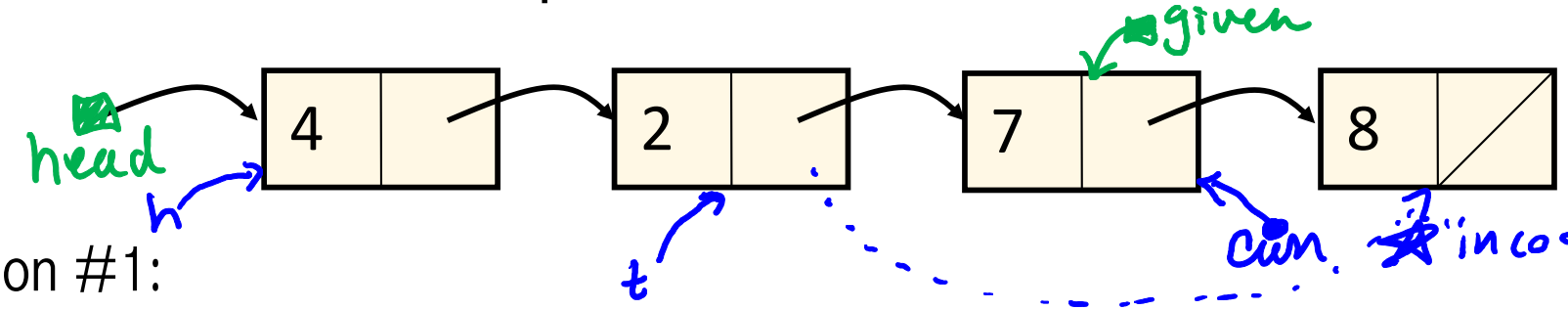

added last after draw it out yourself!

Running time? O(n)

annotations added 1/15.

client code > removeCurrent(head, given);

# Remove node in fixed position (given a pointer to node you wish to remove):




Solution #1:

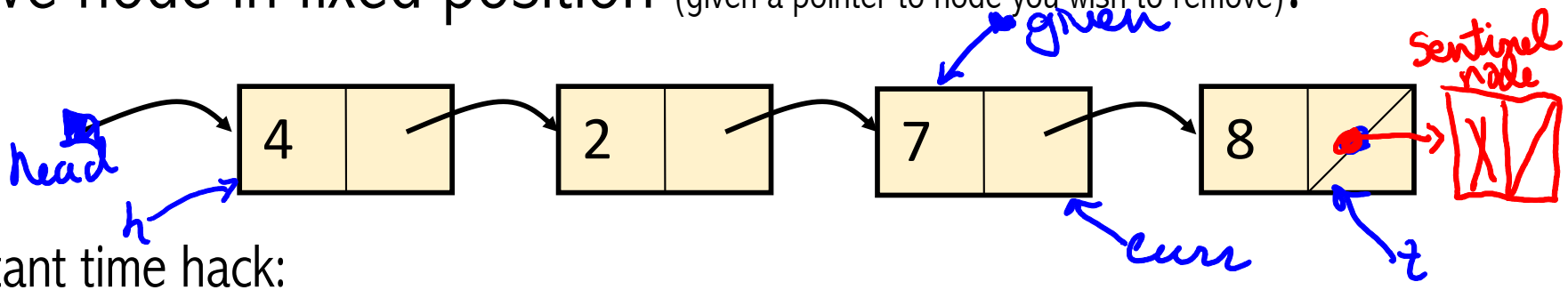
```
void removeCurrent(Node *& h, listNode * curr) {
if (curr != h) {
    Node *t = h;
    while (t->next != curr) t = t->next; // position t to node b4 curr
    t->next = curr->next;
    delete curr;
} else { // curr == h point to front of list
    h = h->next;
    delete curr;
}
}
```

Analysis: worst case, while loop walks down entire list (whose length we denote by n). O(n). Seems like a waste...

annotated on 1/15

client code > removeCurrent(head, given);

# Remove node in fixed position (given a pointer to node you wish to remove):




Constant time hack:

```
void removeCurrent(Node * h, listNode * curr) {
if ( curr->next != NULL) {
   Node * t = curr->next;
   curr->data = t->data;   // OO really?
                           // 0
   curr->next = t->next;
   delete t;
} /* important notes:
      • deleting last node requires either iteration
        OR a sentinel (non-data) node at the tail.
      • copying data can be costly.
*/
```