CompSci 201, L11: Linked List and Pointer Problems

Logistics, Coming up

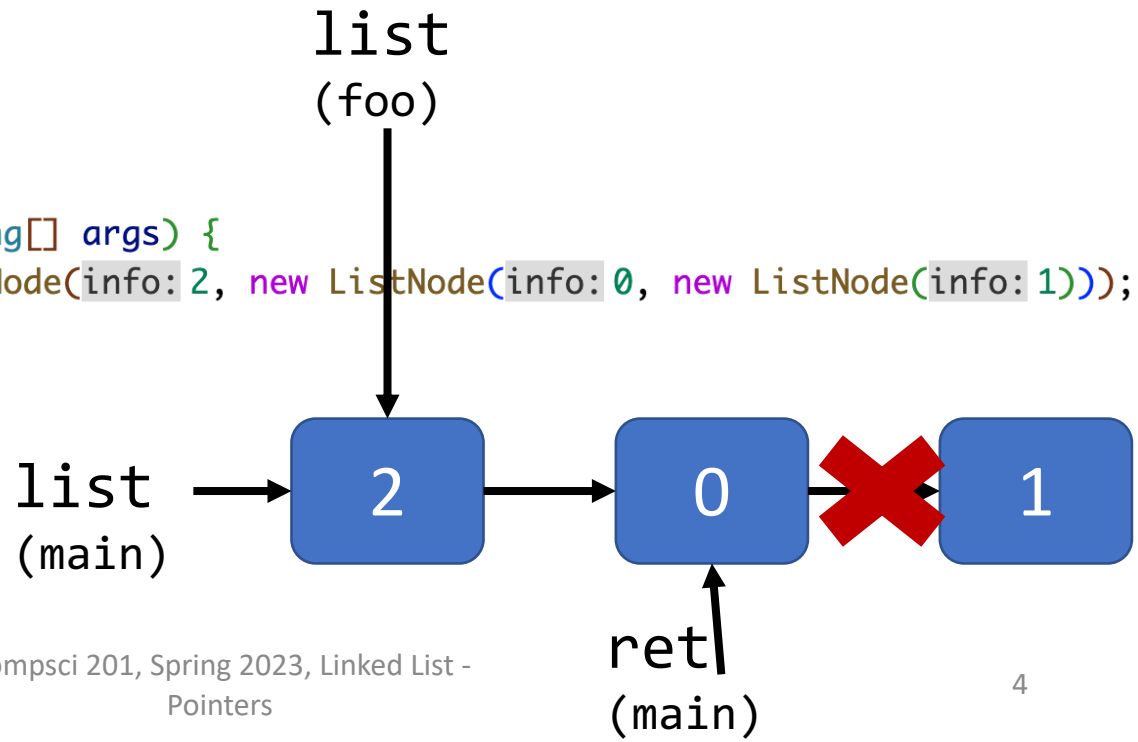
- Monday, 2/20 (today)
  - Project 2: Markov Due
- Wednesday, 2/22
  - APT Quiz 1 due
- Next Monday 2/27
  - Nothing due ☺
  - Work on Project 3: DNA, due the following week

Last time's WOTO Question

What would line 18 print? 0

What would line 19 print? 2, 0

```
 9    public static ListNode foo(ListNode list) {
10        list = list.next;
11        list.next = null;
12        return list;
13    }
14
      Run | Debug
15    public static void main(String[] args) {
16        ListNode list = new ListNode(info: 2, new ListNode(info: 0, new ListNode(info: 1)));
17        ListNode ret = foo(list);
18        printList(ret);
19        printList(list);
20    }
```

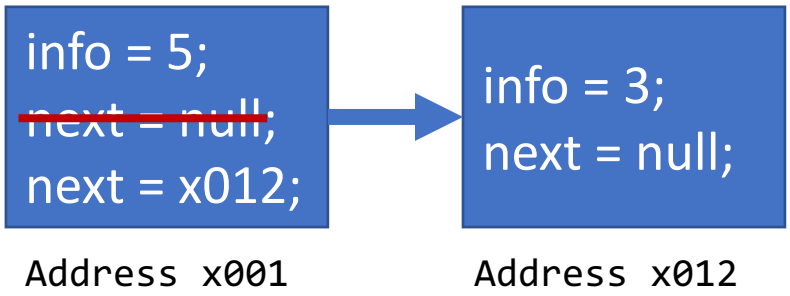


Linked list is a list implemented by linked nodes. What is a node?

- Just a Java object of a class we write, like any other!
- We want to "link" them together, so each node has a *reference* (~pointer, a memory location) to another node.

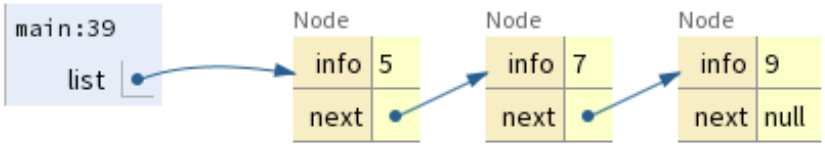
```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int x){
        info = x;
    }
    ListNode(int x, ListNode node){
        info = x;
        next = node;
    }
}
```

```
ListNode first = new ListNode(5);
ListNode second = new ListNode(3);
first.next = second;
```



Creating and traversing a linked list

- `ListNode` class used in APTs, etc.
  - The variable for the "linked list itself" is just a reference to the first `ListNode`




```
ListNode list = new ListNode(5);
list.next = new ListNode(7);
list.next.next = new ListNode(9);
print(list);
```

```
public static void printList(ListNode list) {
    while(list != null) {
        System.out.println(list.info);
        list = list.next;
    }
}
```

While there is a next node…

Print value of current node

Go to next node

DIYLinkedList

Live Coding 

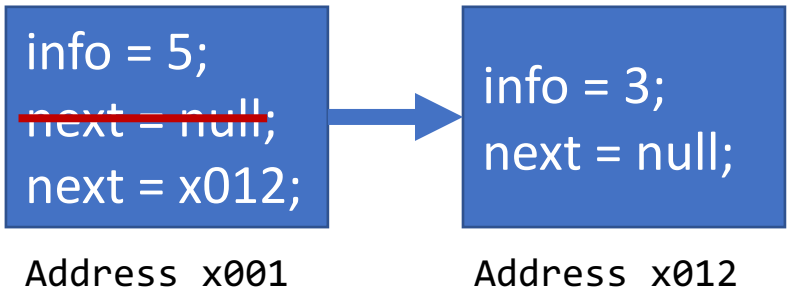
Part 2: Working Directly with List Node objects, algorithmic problem-solving

Linked list is a list implemented by linked nodes. What is a node?

- Just a Java object of a class we write, like any other!
- We want to "link" them together, so each node has a *reference* (~pointer, a memory location) to another node.

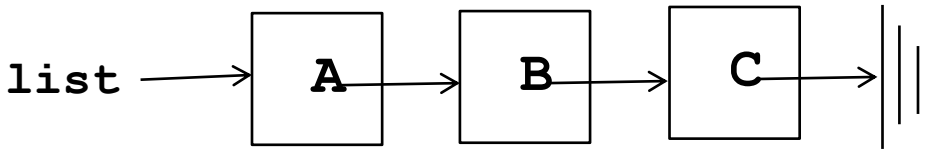
```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int x){
        info = x;
    }
    ListNode(int x, ListNode node){
        info = x;
        next = node;
    }
}
```

```
ListNode first = new ListNode(5);
ListNode second = new ListNode(3);
first.next = second;
```



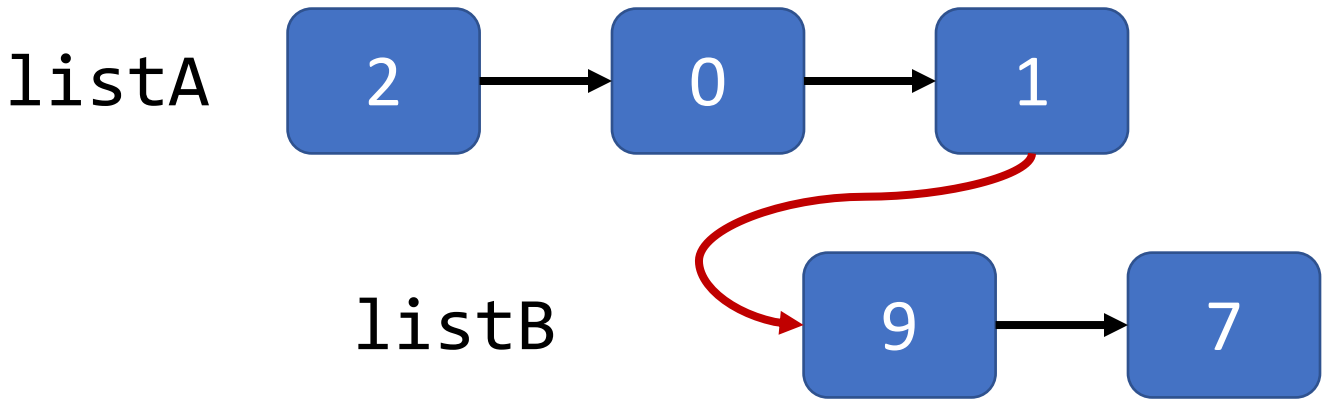
Drawing Pictures

- Visualization is very important: Draw pictures!
  - Try your algorithm/code one step at a time with:
    - 0 nodes
    - 1 node
    - 2 nodes
    - 3 nodes
  - Check boundary conditions
  - Is this pointing to what I think it's pointing to? Check!



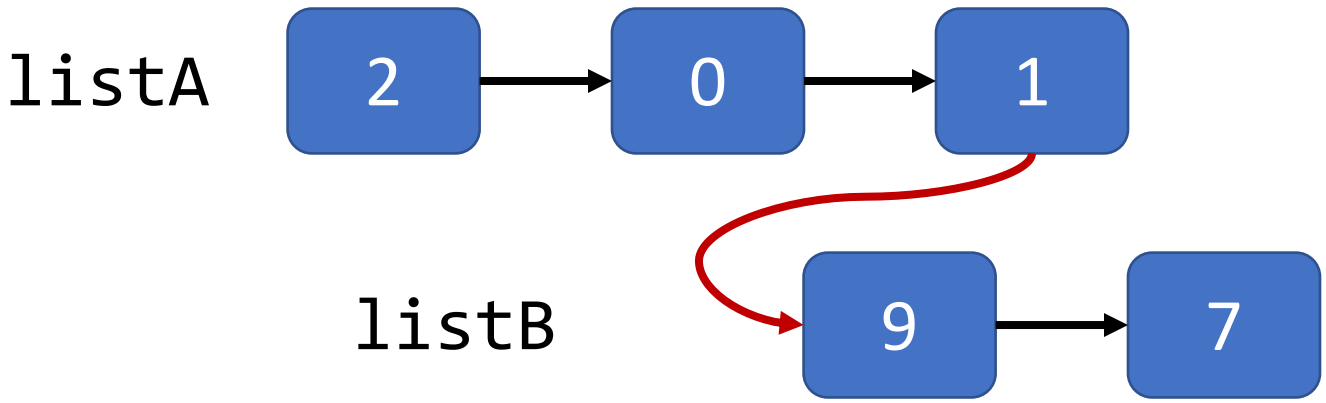
Append linked lists of ListNodes

- Append `listB` to `listA` using…
  - O(1) additional memory,
  - No copying values,
  - Just changing pointers in the input lists.



Append linked lists of ListNodes

- Conceptual algorithmic questions:
  - How to get a reference to the *last* node of `listA`?
  - How to update last node to point to the first node of `listB`?
  - What to return?



How to get a reference to the last node?

Starting with the standard list traversal idiom we know…

```
while (listA != null) {
    listA = listA.next;
}
```

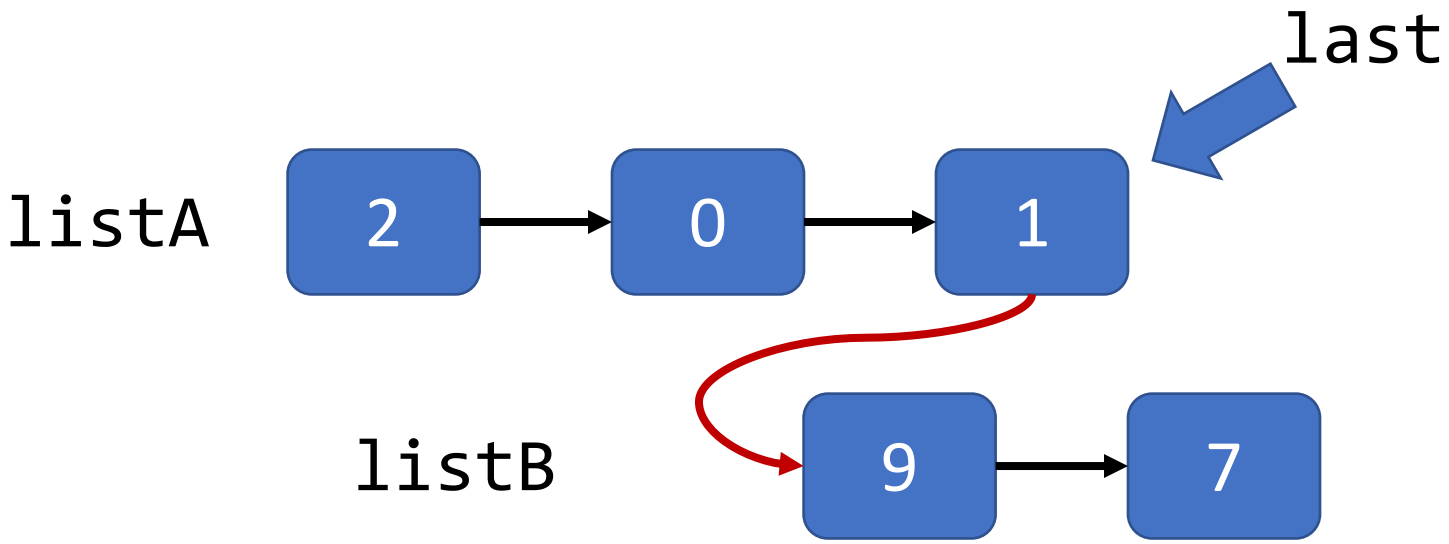
But after exiting this loop, `listA` is just null. Stop one node before…

```
while (listA.next != null) {
    listA = listA.next;
}
```

How to update last node to point to the first node of listB?

Recall: Writing `node.next = otherNode;` makes node → (point to) otherNode.

```
last.next = listB;
```



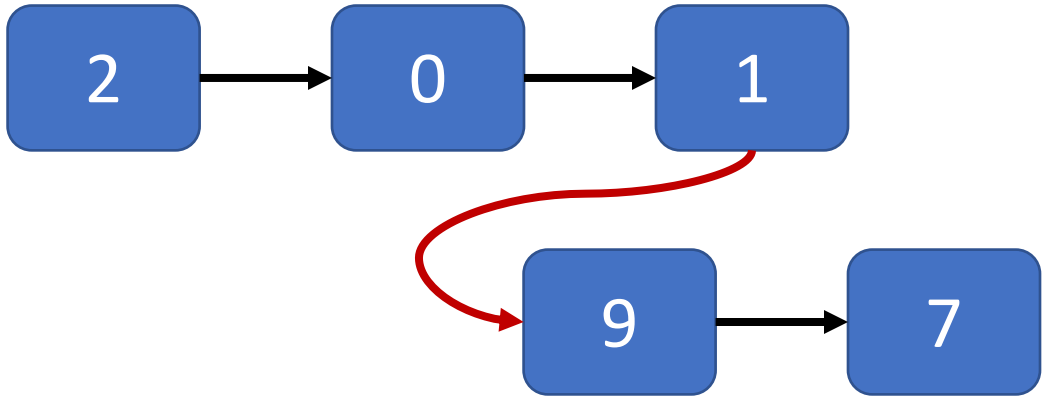
What to return?

If `listB` is appended *to the end* of `listA`, need to return a reference to the first node of `listA`.

```
13      while (listA.next != null) {
14          listA = listA.next;
15      }
16      listA.next = listB;
17      return listA;
```

Correctly changes list in memory, but returns reference to middle

`return`



Append linked lists of ListNodes: Putting it all together

```
12    public static ListNode append(ListNode listA, ListNode listB) {
13        ListNode first = listA;
14        while (listA.next != null) {
15            listA = listA.next;
16        }
17        listA.next = listB;
18        return first;
19    }
```

- Reminding again: Accomplished with O(1) additional memory and without copying any values.
- Not necessarily a lot of lines of code, but…
- easy to get lost without planning and visualization before/while coding.

WOTO

Go to duke.is/yt25t

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

L11-WOTO1-PointerProblems

* Required

* This form will record your name, please fill your name.

1

NetID *

2

Consider the mystery method shown in the code. Suppose listA has n nodes and listB has m nodes. How many new nodes are created in the call to mystery(listA, listB)? *

```
46  public static ListNode mystery(ListNode listA, ListNode listB) {
47      ListNode myList = new ListNode(listA.info);
48      ListNode current = myList;
49      listA = listA.next;
50
51      while (listA != null) {
52          current.next = new ListNode(listA.info);
53          current = current.next;
```

```
54          listA = listA.next;
55      }
56
57      while (listB != null) {
58          current.next = new ListNode(listB.info);
59          current = current.next;
60          listB = listB.next;
61      }
62
63      return myList;
64  }
```

- ○ 0
- ○ n
- ○ m
- ✓ n+m
- ○ n+m+1
- ○ n-m
- ○ nm

3

Same mystery method shown in the code. Suppose listA has n nodes and listB has m nodes. The runtime complexity of mystery is... *

```
46  public static ListNode mystery(ListNode listA, ListNode listB) {
47      ListNode myList = new ListNode(listA.info);
48      ListNode current = myList;
49      listA = listA.next;
50
51      while (listA != null) {
52          current.next = new ListNode(listA.info);
53          current = current.next;
54          listA = listA.next;
55      }
56
57      while (listB != null) {
58          current.next = new ListNode(listB.info);
59          current = current.next;
60          listB = listB.next;
61      }
62
63      return myList;
64  }
```

- ◯ O(1)
- ◯ O(n)
- ◯ O(m)
- ◯ O(n+m) ✓
- ◯ O(nm)

4

Same mystery method. Suppose listA is initially [2, 0, 1] and listB is initially [4, -1]. What will myList be when returned? *

```
46  public static ListNode mystery(ListNode listA, ListNode listB) {
47      ListNode myList = new ListNode(listA.info);
48      ListNode current = myList;
49      listA = listA.next;
50
51      while (listA != null) {
52          current.next = new ListNode(listA.info);
53          current = current.next;
54          listA = listA.next;
55      }
56
57      while (listB != null) {
58          current.next = new ListNode(listB.info);
59          current = current.next;
60          listB = listB.next;
61      }
62
63      return myList;
64  }
```

- ◯ [] (empty)
- ◯ [2, 0, 1]
- ◯ [4, -1]
- ◯ [2, 0, 1, 4, -1] ✓
- ◯ [-1, 0, 1, 2, 4]

5

Same mystery method. Suppose we have a program that invokes it as follows:

ListNode listA = ... // creates a linked list
ListNode listB = ... // creates another linked list
ListNode newList = mystery(listA, listB)
// More code that uses listA and listB

In the additional code using listA and listB after the creation of newList, are listA and listB any different than before the method call? *

```
46  public static ListNode mystery(ListNode listA, ListNode listB) {
47      ListNode myList = new ListNode(listA.info);
48      ListNode current = myList;
49      listA = listA.next;
50
51      while (listA != null) {
52          current.next = new ListNode(listA.info);
53          current = current.next;
54          listA = listA.next;
55      }
56
57      while (listB != null) {
58          current.next = new ListNode(listB.info);
59          current = current.next;
60          listB = listB.next;
61      }
62
63      return myList;
64  }
```

- ◯ Yes, different values and different node objects
- ◯ Different values but the same node objects
- ◯ Different node objects but the same values
- ◯ No, same values and same node objects

6

We looked at this append method in lecture. If listA has n nodes and listB has m nodes, the runtime complexity of append is... *

```
12  public static ListNode append(ListNode listA, ListNode listB) {
13      ListNode first = listA;
14      while (listA.next != null) {
15          listA = listA.next;
16      }
17      listA.next = listB;
18      return first;
19  }
```

- ◯ O(1)
- ✓ O(n)
- ◯ O(m)
- ◯ O(n+m)
- ◯ O(nm)

7

Same append method. Suppose we have a program that invokes it as follows:

ListNode listA = ... // creates a linked list
ListNode listB = ... // creates another linked list
ListNode newList = append(listA, listB)
// More code that uses listA and listB

In the additional code using listA and listB after the creation of newList, what is the relationship between listA and newList? *

```
12  public static ListNode append(ListNode listA, ListNode listB) {
13      ListNode first = listA;
14      while (listA.next != null) {
15          listA = listA.next;
16      }
17      listA.next = listB;
18      return first;
19  }
```

- ◯ listA refers to the same object as newList
- ◯ listA and newList have the same values but refer to different objects
- ◯ No relationship, different values and different objects

Canonical Linked List Problem

- How do we reverse nodes in a linked list (without creating a new list)?
  - Go from A->B->C to C->B->A
  - Typical interview style question
  - https://leetcode.com/problems/reverse-linked-list/
  - https://www.hackerrank.com/challenges/reverse-a-linked-list