LINKED LIST

BME 121 2016

Jeff Luo

Topics

- The Linked List Data Structure
  - Prepend
  - InsertAt
  - InsertInOrder
  - RemoveHead
  - RemoveTail
  - RemoveAt
  - RemoveMin
  - RemoveMax

- Open up LinkedListPractice.cs on OneDrive to work on the code!

Comparing Strings

Eg given:

string x = “grumpy”;

string y = “cat”;

To compare strings according to alphabetical order, use:

string.Compare(x, y)

X vs Y alphabetically	Compare(x, y) returns
X < Y	A number < 0
X == Y	0
X > Y	A number > 0

Linked List - Prepend

- Prepending a node to the tail of a list (2 cases):

  Case 1: no nodes in the linked list

  → Create a new node at head

  → Set tail to also point to this node

  → count++

  Case 2: 1+ nodes

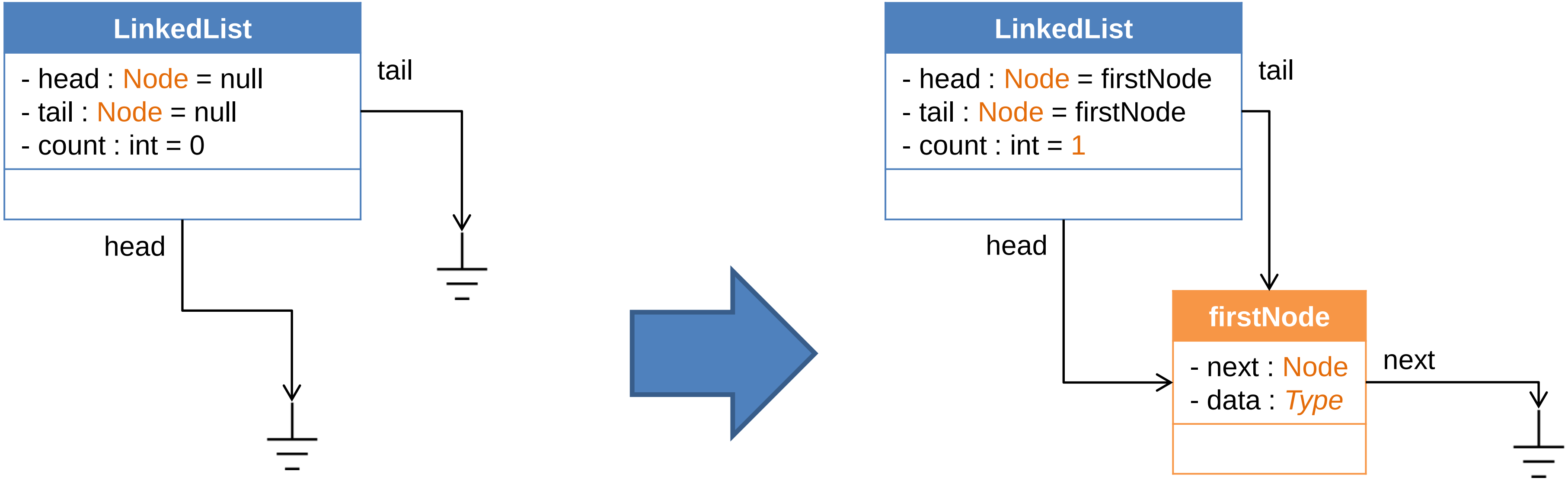
  → Create a new node

  → Set this node’s Next to the head node

  → Update head to point to this new node

  → count++

Prepending to a Linked List – Case 1



Empty Linked List

LinkedList

- head : Node = null
- tail : Node = null
- count : int = 0

Linked List with 1 Node

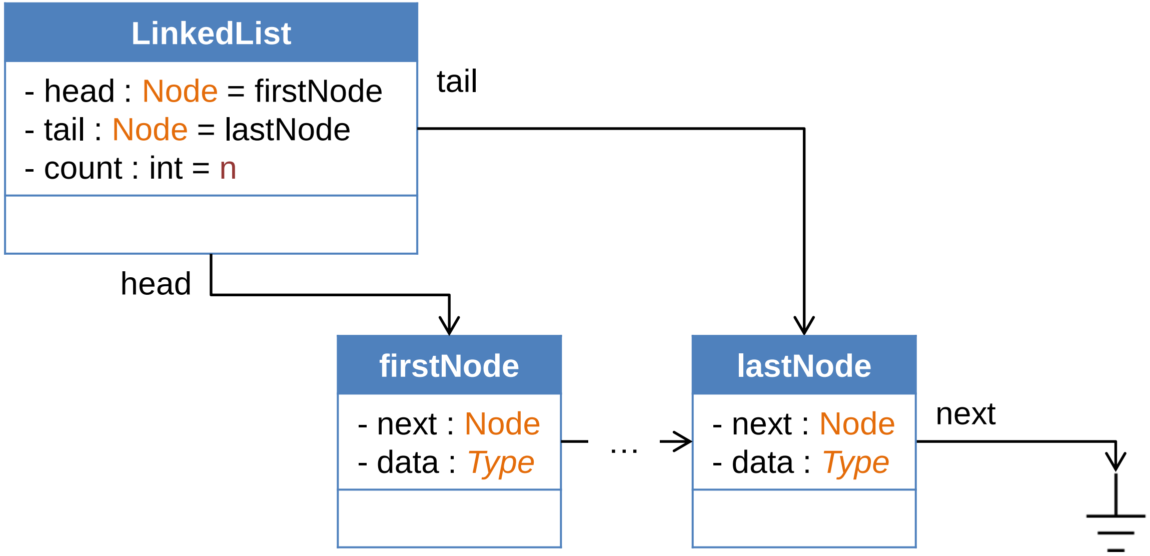
LinkedList

- head : Node = firstNode
- tail : Node = firstNode
- count : int = 1

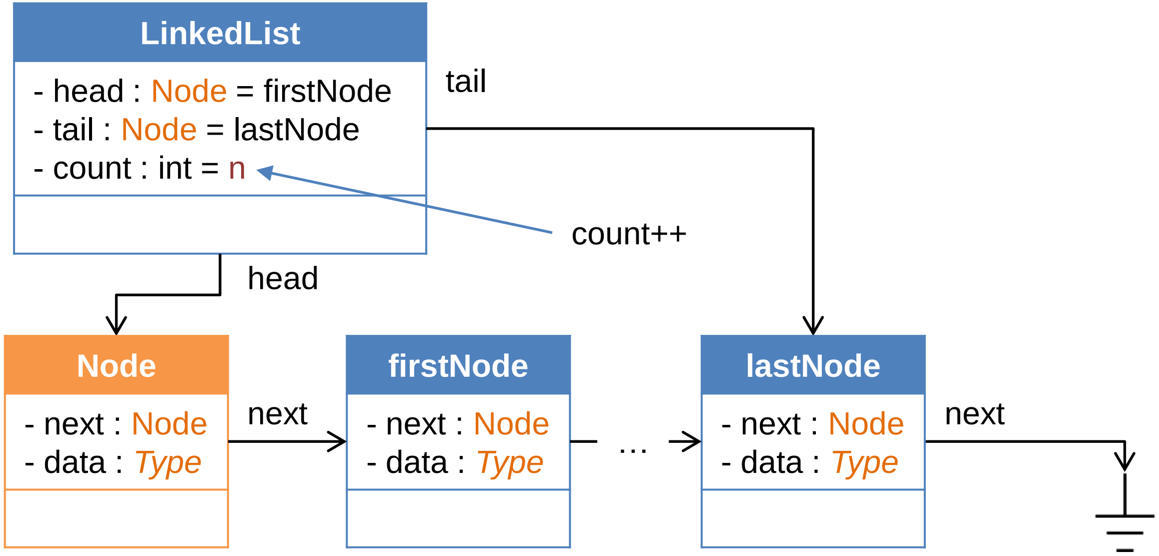
firstNode

- next : Node
- data : *Type*

Prepending to a Linked List – Case 2 (Before)

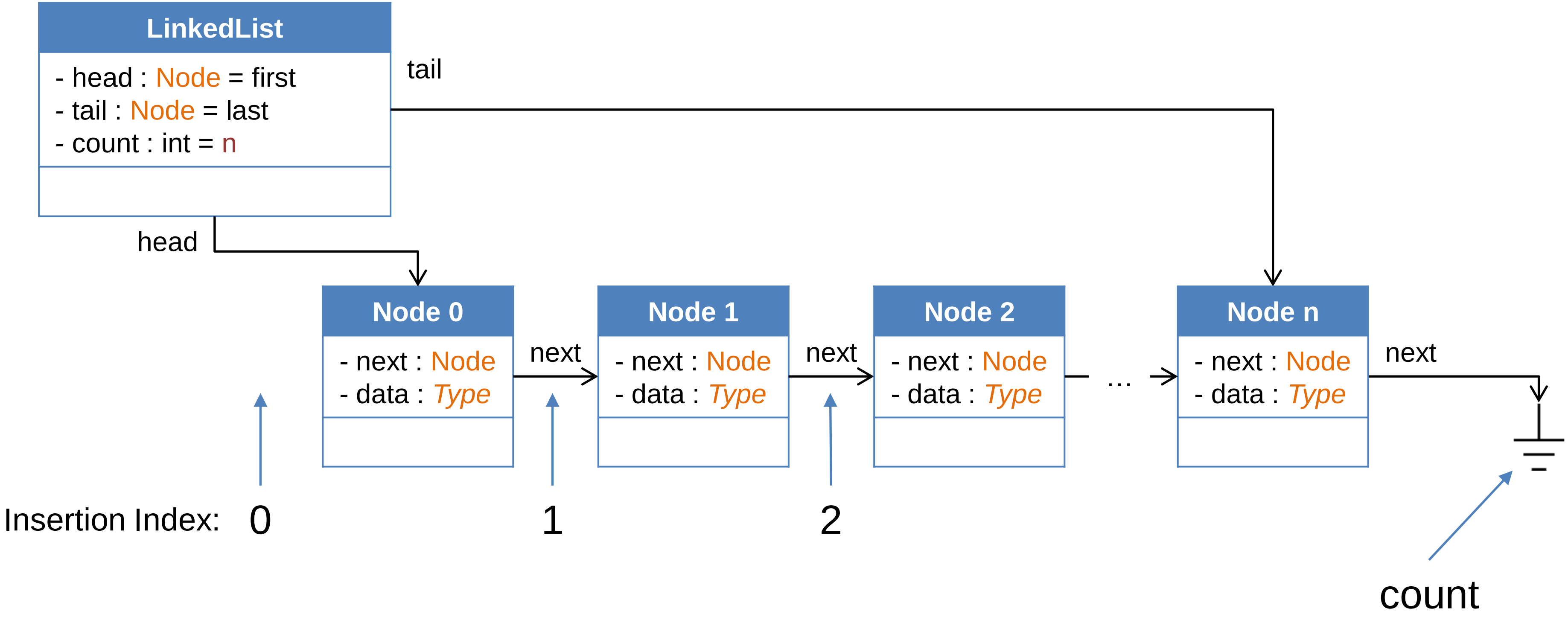


Prepending to a Linked List – Case 2 (After)



Linked List – InsertAt(int index, string value)

Inserting a node at some index:



Index convention: insert at X means insert just before node X

Linked List – InsertAt(int index, string value)

- Inserting a node at some index:

First check to make sure the index is >=0 and <= count, if so, 3 cases:

Case 1: index == 0

→ Prepend the value

Case 2: index == count

→ Append the value

Case 3: index is between 0 and count

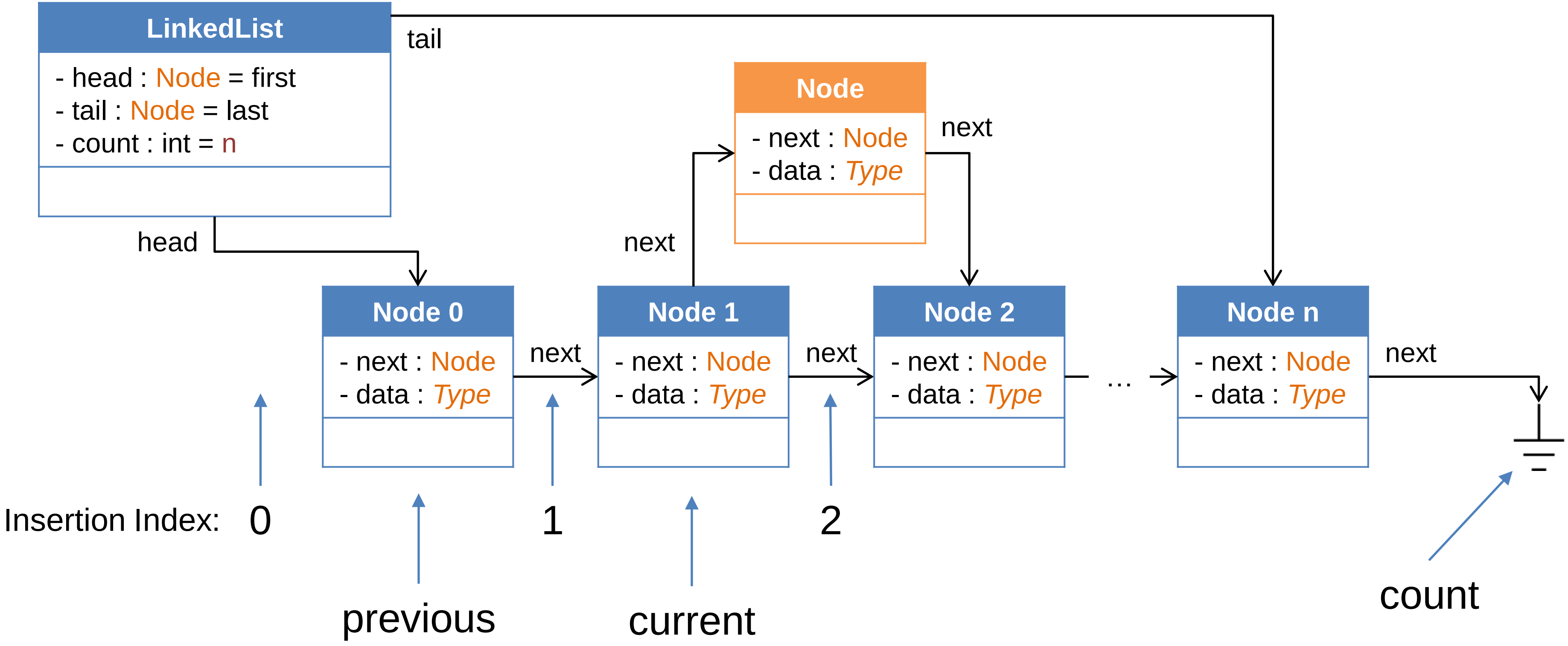
→ Use two references called previous and current, start previous at head and current at head.Next, and move them together down the linked list until we are at the index

→ Create a new node between previous and current

→ count++

Linked List – InsertAt(int index, string value)

Eg: InsertAt(2, “cool”)



Linked List – InsertInOrder(string value)

- Assumes that the existing linked list is sorted (eg ascendingly).
- Inserting a node in a sorted order (eg ascending), 2 major cases:

Case 1: no nodes in the linked list

→ Prepend the value

Case 2: 1+ nodes

Subcase 1: if value <= head value, prepend the value

Subcase 2: if value >= tail value, append the value

Subcase 3: → Use two references called previous and current, start previous at head and current at head.Next, and move them together down the linked list until **previous value < value <= current value**

→ Create a new node between previous and current

→ count++

Linked List – RemoveHead

- Remove head node and return the data value stored in that node (3 cases):

  Case 1: no nodes in the linked list

  → Return null

  Case 2: 1 node

  → Create a temporary reference to the head node

  → Set head = null, tail = null, count = 0

  → Return the data stored in the old head node

  Case 3: 2+ nodes

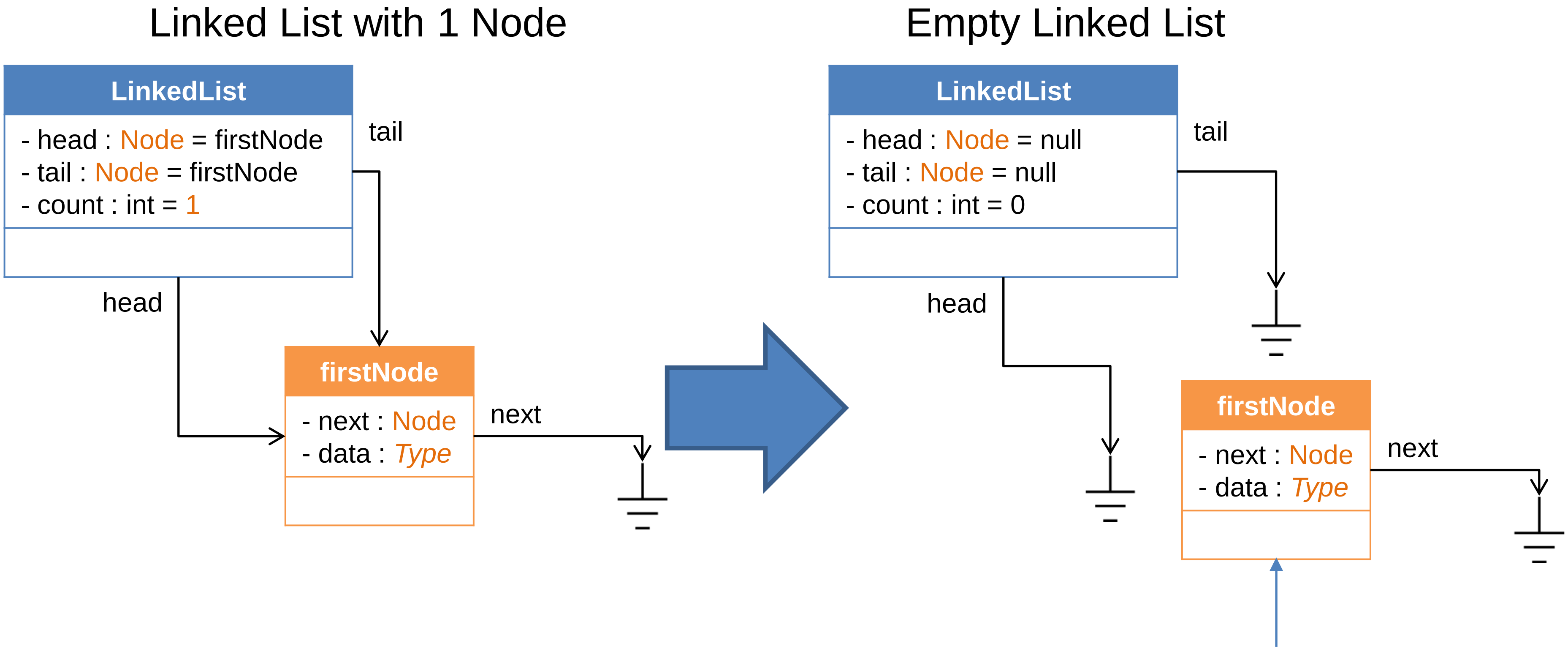
  → Create a temporary reference to the head node

  → Set head reference to point to the node after head node

  → Set the old head node’s Next to null

  → count--, and return the data stored in the old head node

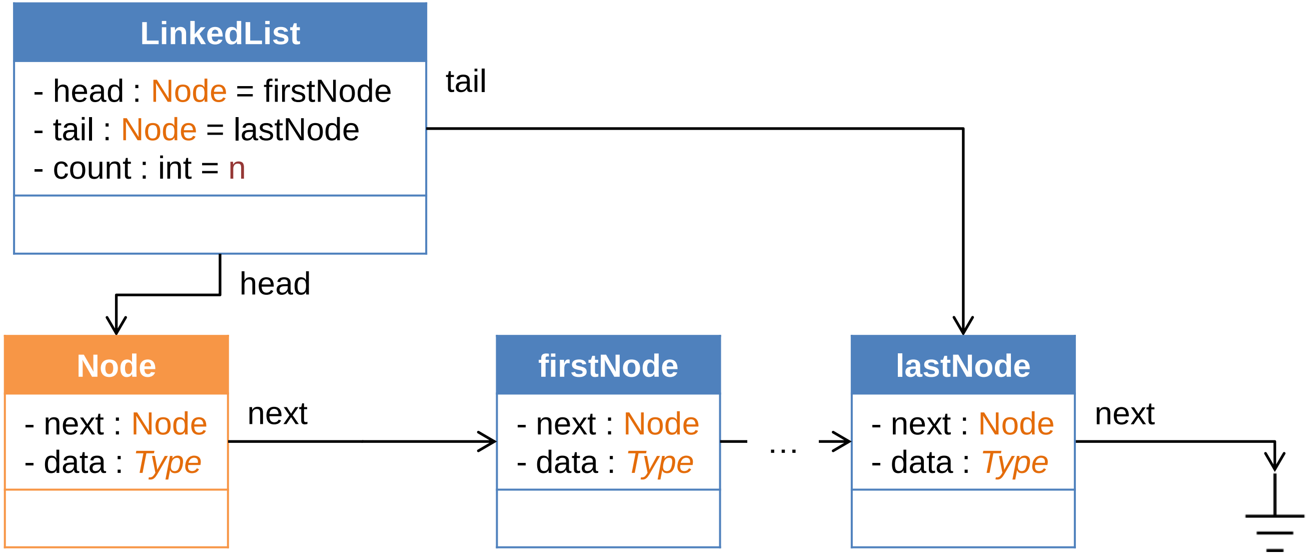
Removing Head Node – Case 2



temp

return temp.GetData();

Removing Head Node – Case 3 (Before)



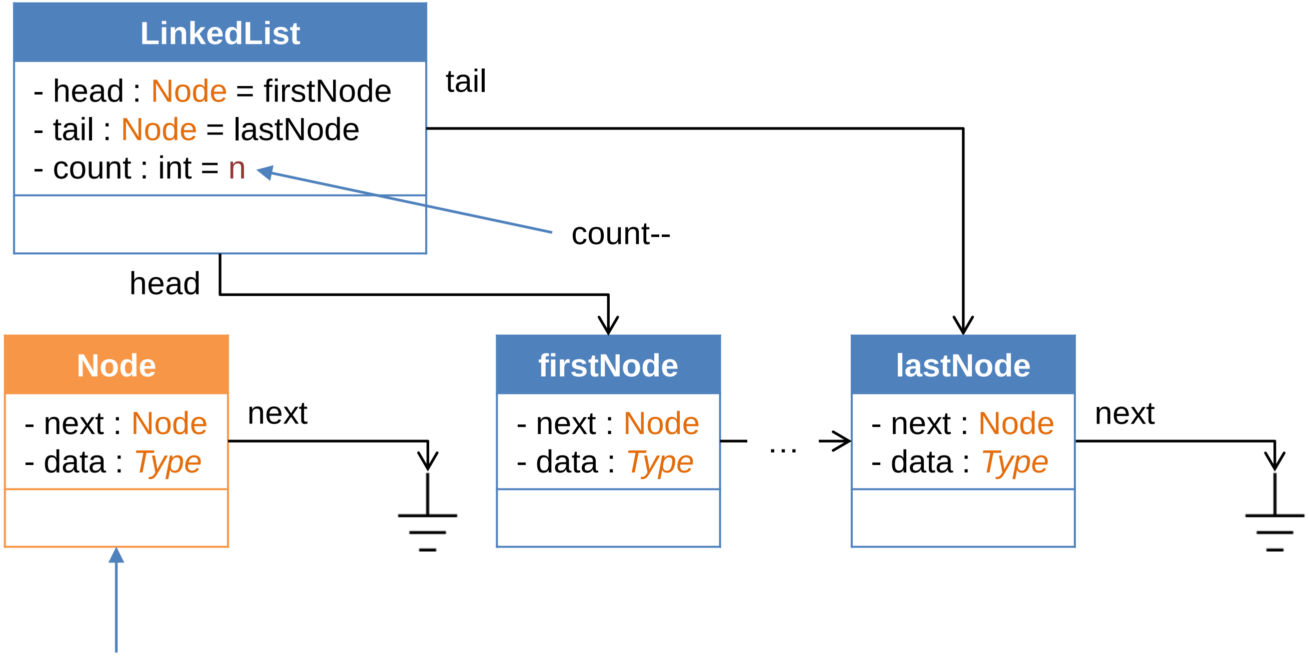
LinkedList
- head : Node = firstNode
- tail : Node = lastNode
- count : int = n

Node
- next : Node
- data : *Type*

firstNode
- next : Node
- data : *Type*

lastNode
- next : Node
- data : *Type*

Removing Head Node – Case 3 (After)



LinkedList

- head : Node = firstNode
- tail : Node = lastNode
- count : int = n

count--

Node

- next : Node
- data : *Type*

firstNode

- next : Node
- data : *Type*

lastNode

- next : Node
- data : *Type*

temp

return temp.GetData();

Linked List – RemoveTail

- Remove tail node and return the data value stored in that node (3 cases):

  Case 1: no nodes in the linked list

  → Return null

  Case 2: 1 node

  → Create a temporary reference to the tail node

  → Set head = null, tail = null, count = 0

  → Return the data stored in the old tail node

  Case 3: 2+ nodes

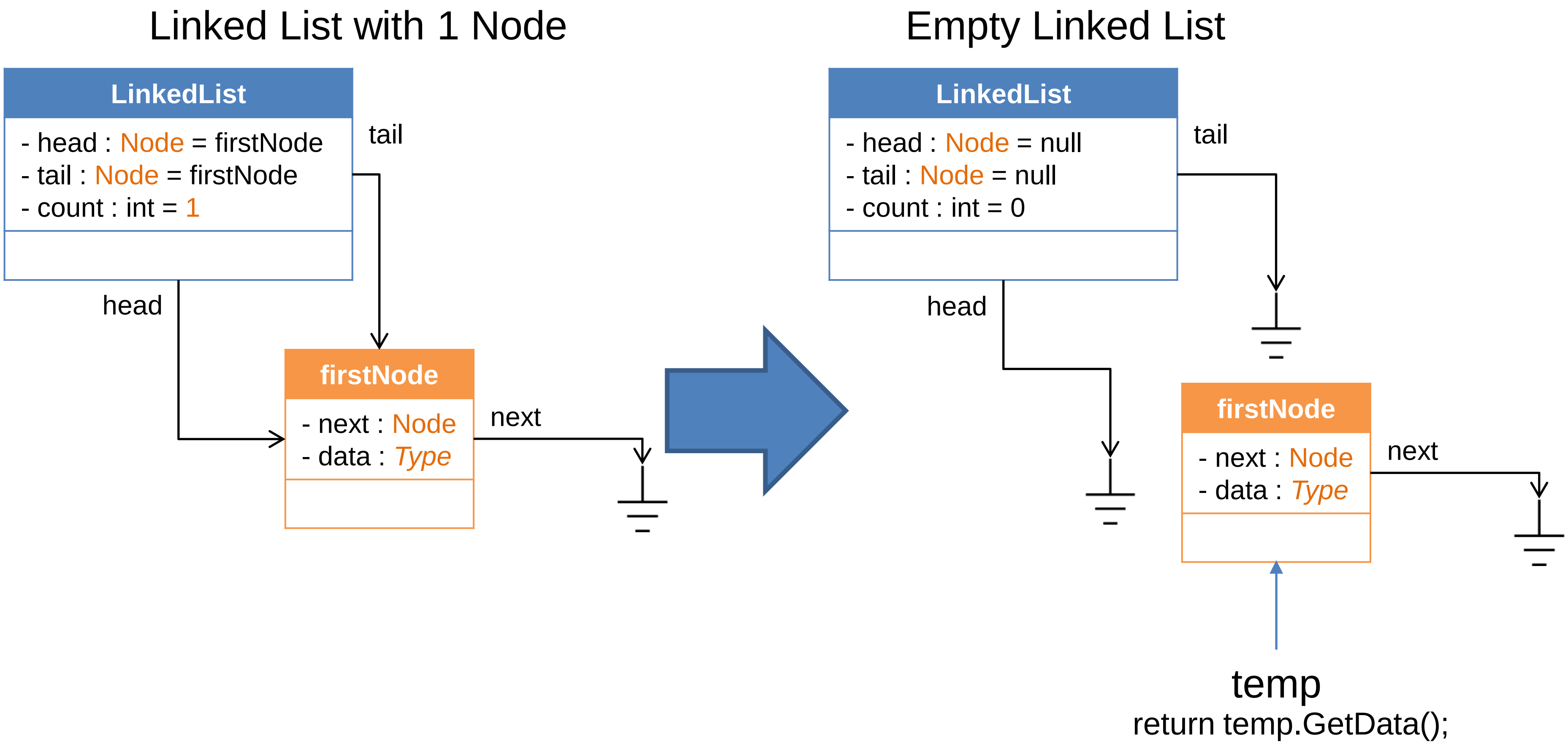
  → Use two references called previous and current, start previous at head and current at head.Next, and move them together down the linked list until **current == tail**

  → Set tail reference to point to the previous node

  → Set the new tail node's Next = null

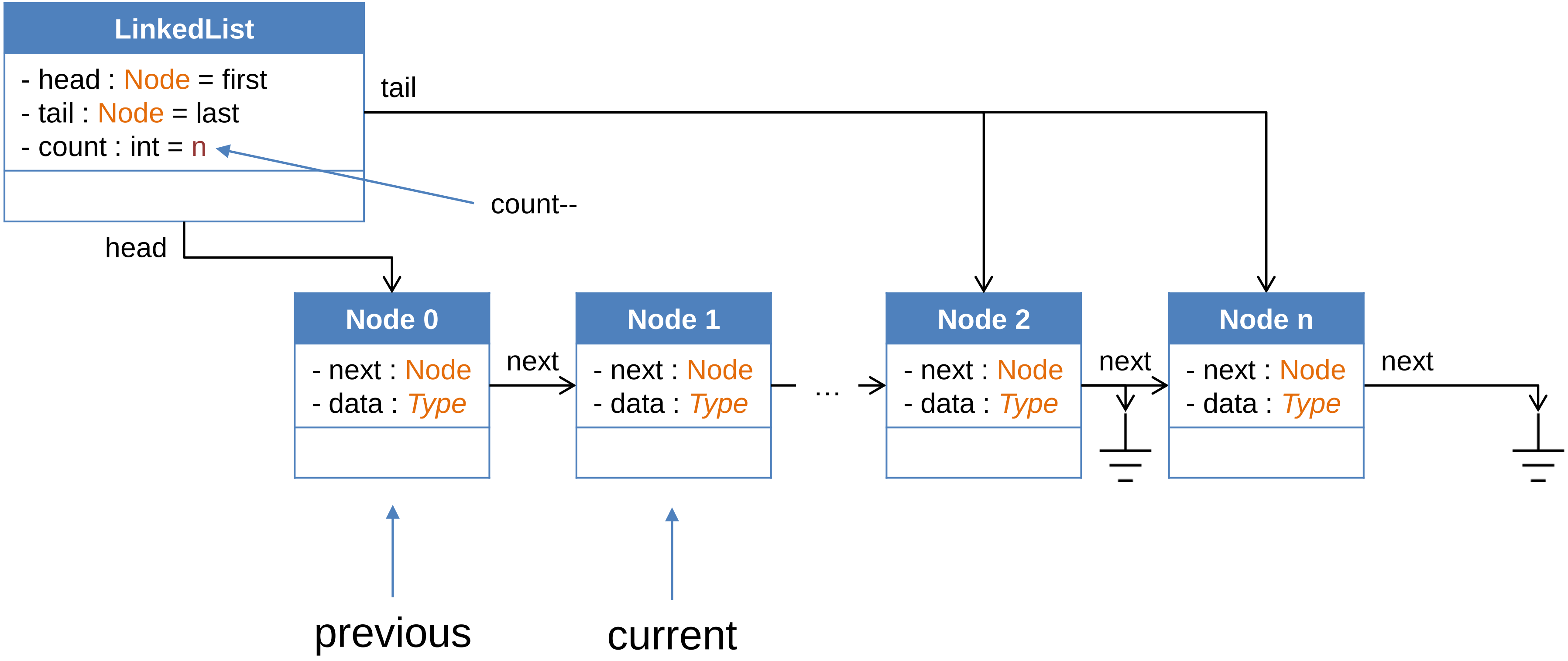
  → count--, and return the data stored in the old tail node

Removing Tail Node – Case 2



return temp.GetData();

Removing Tail Node – Case 3



return current.GetData();

Linked List – RemoveAt(int index)

- Removing a node at some index:

First check to make sure the index is >=0 and <= count - 1, if so, 3 cases:

Case 1: index == 0

→ return RemoveHead()

Case 2: index == count - 1

→ return RemoveTail()

Case 3: index is between 0 and count - 1

→ Use two references called previous and current, start previous at head and current at head.Next, and move them together down the linked list until current points to the node we want to remove

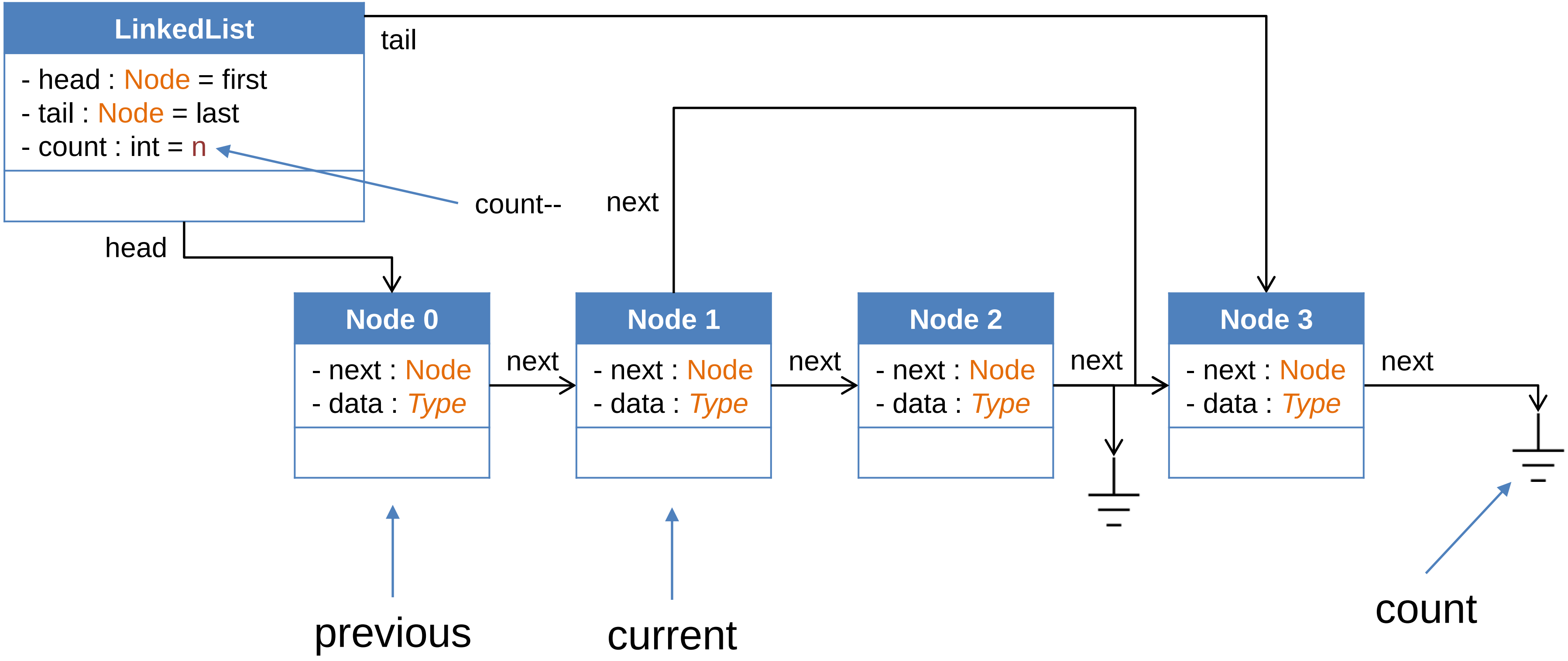
→ Unlink current node by setting previous node's Next to point to the node after current node

→ Also set current.Next = null

→ count--, and return the data stored in the node referenced by current

Linked List – RemoveAt(int index)

Eg: RemoveAt(2)



return current.GetData();

Linked List – RemoveMin

- Removing the node with the minimum value (2 cases):

  Case 1: no nodes in the linked list

  → return null

  Case 2: 1+ nodes

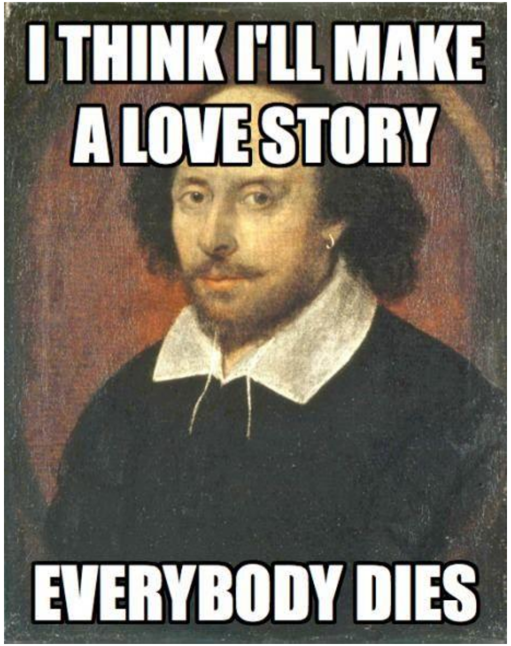
  → Use forward brute force search to find the index of the node with the min value:

  ```
  → Assume head node's value is min value to start
  → Create int x and int index, both start at 0
  → Create a temporary reference starting at head node
  → While temp reference != null
      → Compare each node's value to minValue, if it is lower:
              → index = x  and  minValue = the node's value
      → x++  and temp = temp.Next
  → After the loop, return RemoveAt(index)
  ```

Linked List – RemoveMax

- Practically identical to RemoveMin, except for the comparison and update of maxValue

William Shakespeare



Romeo and Juliet
(written 1591-1595)

William Shakespeare – Words

- How many words did Shakespeare write in his life?
  - Can solve this by counting the words

- How many unique words did Shakespeare write in his life?
  - Use a Linked List!
  - See ShakespeareLL.cs