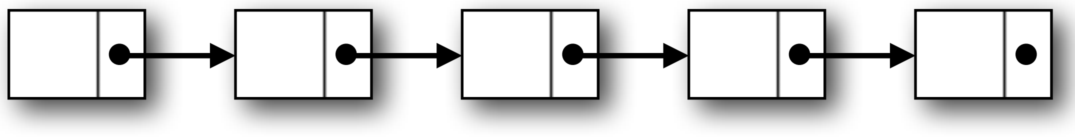
Linked Lists

CS 1331 Extra Credit Homework



Overview

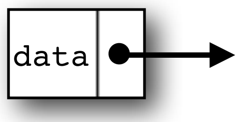
This homework is an opportunity to make up points lost on previous assignments. It is optional, but highly recommended that you complete it, as it will help prepare you for CS 1332. By the time you've completed this assignment, you should have a basic grasp of the linked list data structure.

Linked List

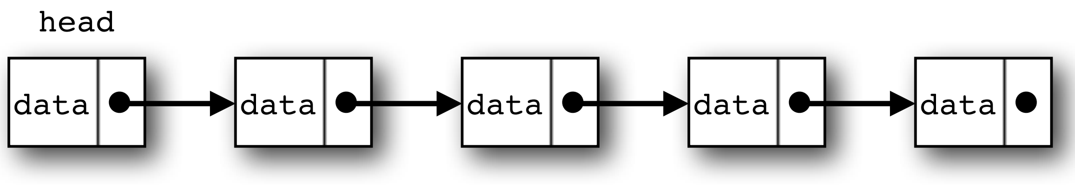
A linked list is a data structure. Like an array, a linked list holds a series of objects that are logically grouped together. Linked lists differ from arrays in that they can hold an arbitrary amount of data without the overhead of having to create larger arrays and copying already existing elements into them. Additionally, the design of linked lists implies an order to elements placed into them, as we will see in the example below.

Linked lists are made up of a series of nodes, which are linked together to form a list, hence their name. A node is nothing more that a reference to an object of the type the list should hold and a reference to the next node.

A Node:

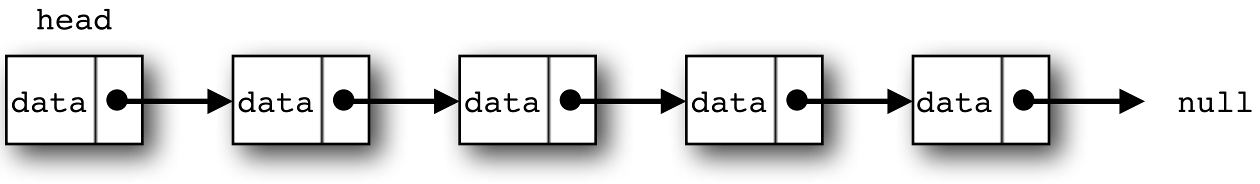


By keeping a reference to the node at the front, or "head", of the list, and linking it to subsequently linked nodes, we create a linked list.



Note that the nodes don't all have to contain the same data, just data of the same *type*.

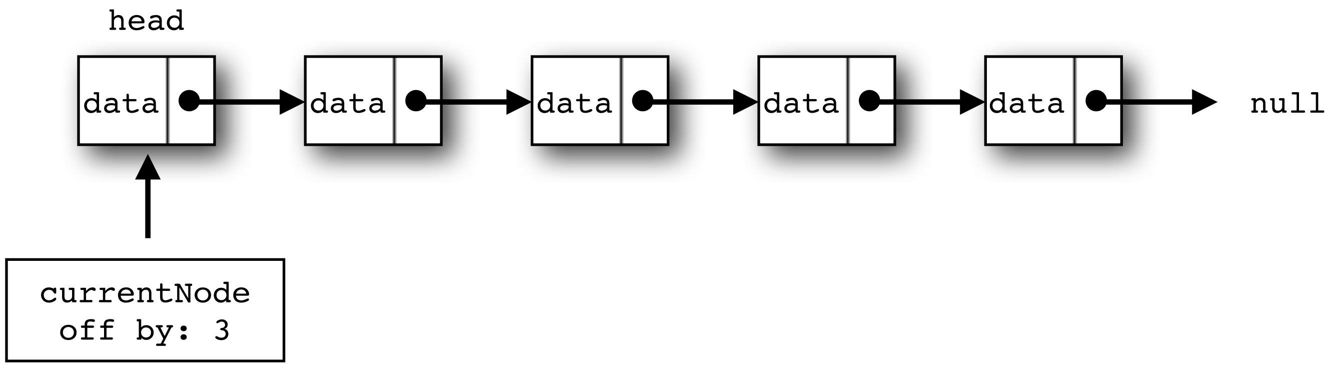
You may be wondering what the node at the end of the list is linked to. There are a variety of ways to handle ending the list, but for this assignment the last node in the list will be linked to `null`.



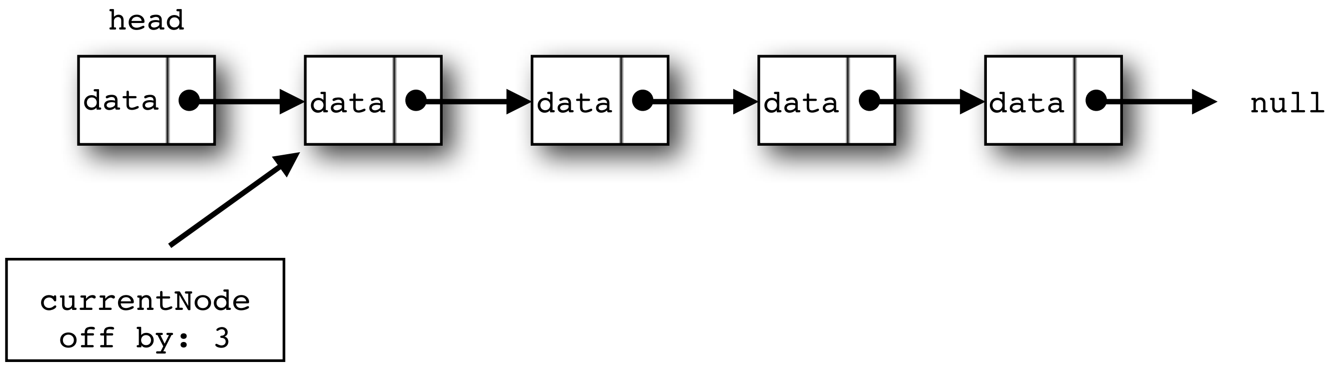
Traversing the List

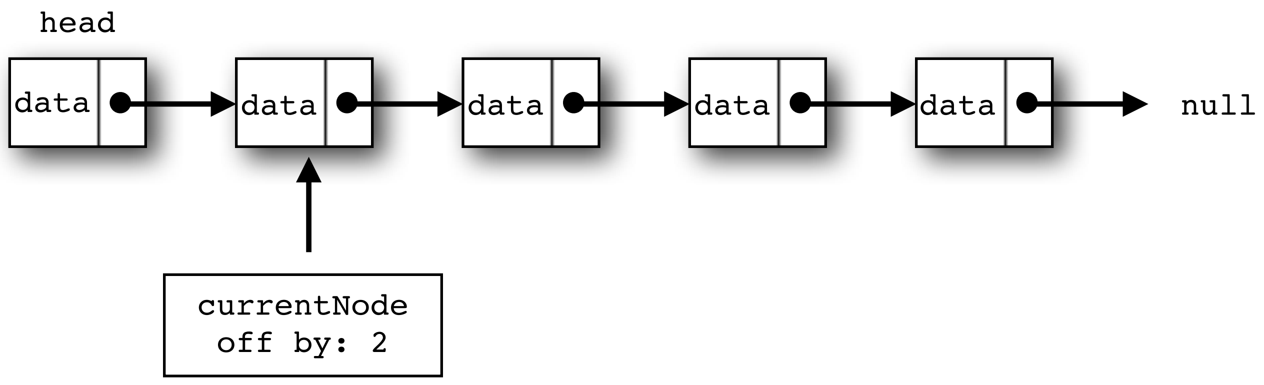
Traversing a linked list is simple. As the linked list only has a reference to the head, we start with that node. To reach a particular ordinal node, we walk from node to node until the node with the position we were looking for is reached (or we find the end of the list).

Consider the scenario in which we want to find the data in the 3rd node (remember that we start counting at 0) of the list. We inspect the nodes one at a time, the currently inspected node starting at the head of the list.

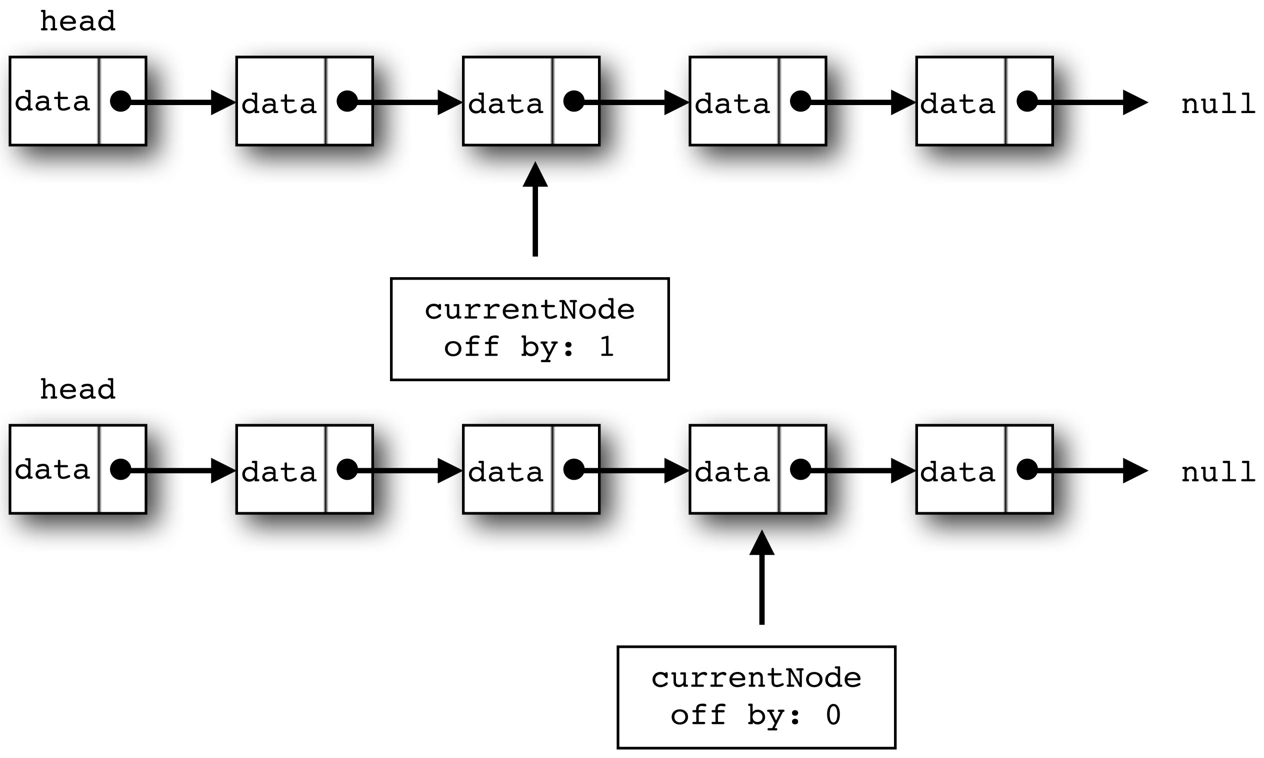


This is obviously not the node we want, so we move to the next node. This is accomplished by setting the currentNode equal to the next node in the list.





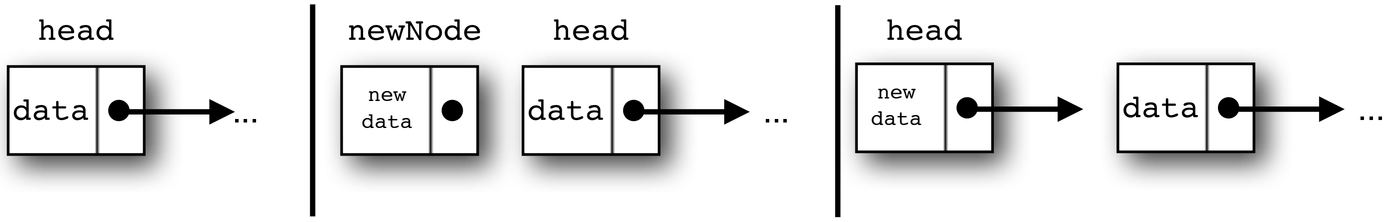
We repeat this until we've reached the 3rd node.



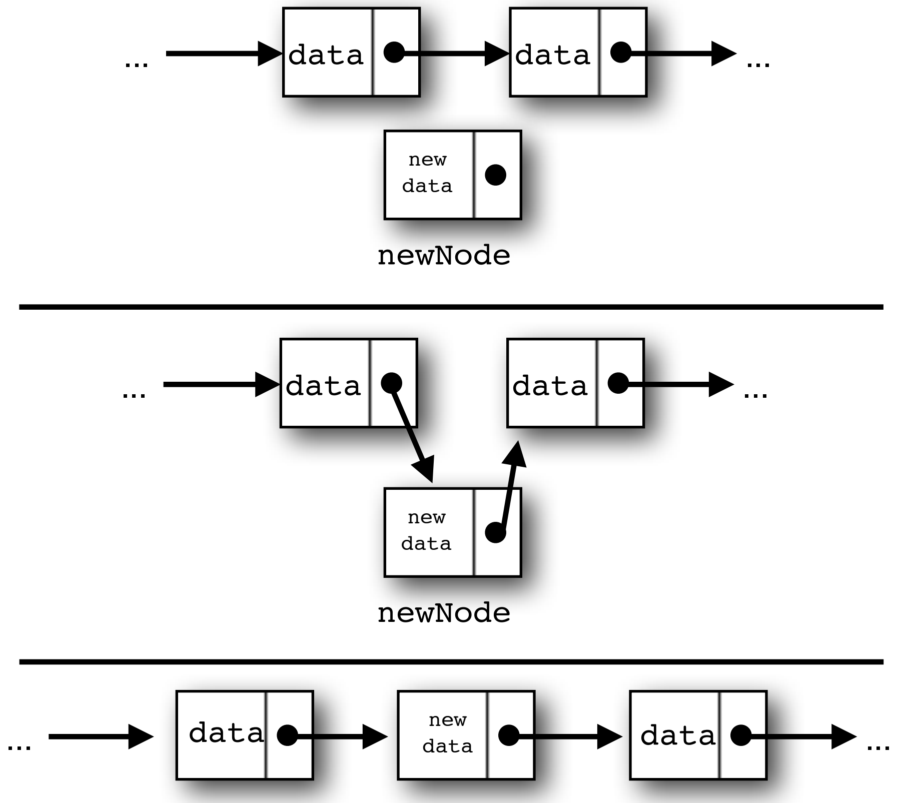
Having found our node, we return its data.

Adding to the List

Adding to the front of the list is the most simple case. For a list of any length, adding the front of the list involves creating a new node, setting its next to be the current head, and then setting the head reference to the new node.



To insert a node at an arbitrary position in the list, first traverse to the position the node should be. Then set the new node's next to be the previous node's next and the previous node's next to the new node.



If the next node in the list is null, then the new node's next will be null.

Removing from the List

Removing from the list is exactly the opposite of adding. The node previous to the node being removed has its next set to be the node after the removed node.

Your Assignment, Should You Choose to Accept It

We have provided a skeleton class with empty methods. You are to fill in the methods with code as specified by the javadocs for each method. Additionally, create a Test class with a main() method, in which you must test all of the methods of the StringLinkedList class.

This assignment is due Monday, December 10th (the first day of finals week).

Deliverables:

- StringLinkedList.java
- Test.java