Linked Lists

Christopher Simpkins
chris.simpkins@gatech.edu

Linked Lists

- Dynamic data structures
- Singly linked lists
- Generic linked lists
- Doubly linked lists

Dynamic Data Structures

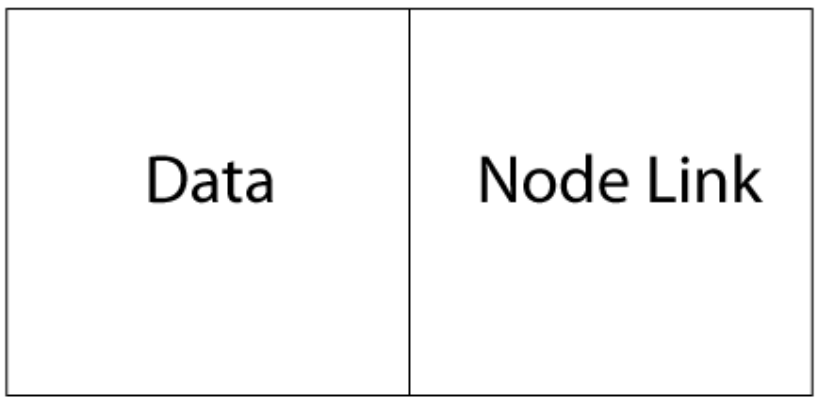
- `ArrayList` was our first dynamically-allocated data structure.
  - `ArrayList` is a hybrid static and dynamically allocated data structure.
  - `ArrayList` automatically allocates a larger backing (static) array when the capacity of its current backing array is exceeded.
- In a purely dynamic data structure, storage for each new element is allocated when the element is added.
- A purely dynamically allocated data structure is (slightly) more space efficient, but slower because heap allocation occurs every time an element is added.

Linked Data Structures

The core concept in a linked data structures is the node.

- A data structure is a collection of nodes linked in a particular way.
- A node holds a data item and links to other nodes.
- The nature of the links determines the kind of data structure, e.g., singly linked list, doubly linked list, binary tree, etc.

Here is a depiction of a node for a singly linked list and code that implements such a node (public members for brevity)



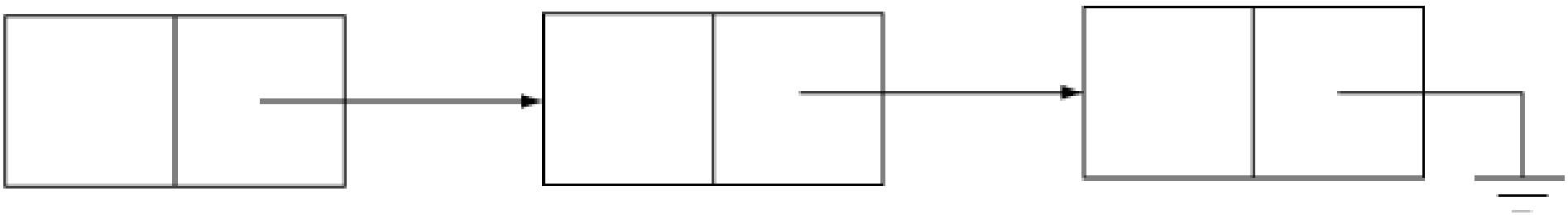

```
private class Node {
  public Object data;
  public Node next;

  public Node(Object data, Node next) {
    this.data = data;
    this.next = next;
  }
}
```

Singly Linked Lists

A singly linked list

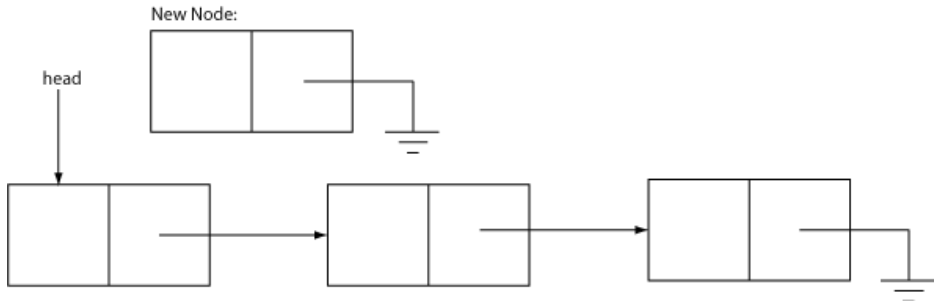
- Contains a pointer to a “head” node (`null` if empty).
- The head node’s `next` points to the second node, the second node’s `next` points to the third node, and so on.
- The `next` reference of the last node is `null`

Here’s a depiction of the nodes in a singly linked list with three elements:

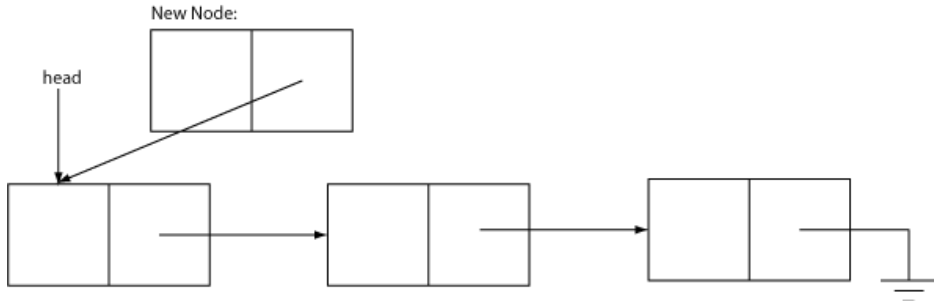


Adding Elements to a Singly Linked List

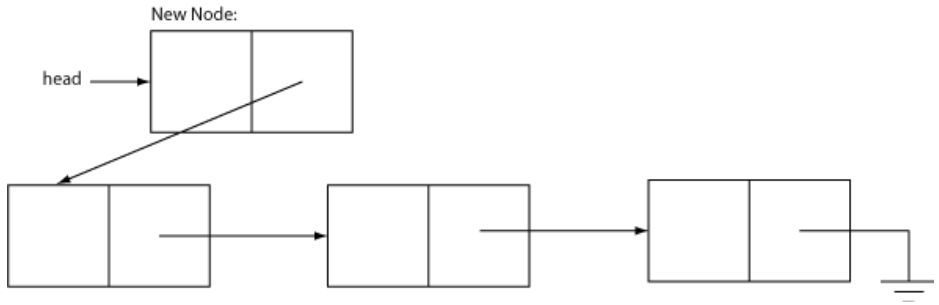
1. Create a new `Node`.



2. Set the `next` reference of the new `Node` to the current `head`.



3. Set the `head` reference to the new `Node`



See LinkedList.java for the code.

Finding an Item in a Linked List

An algorithm for finding an item in a linked list:

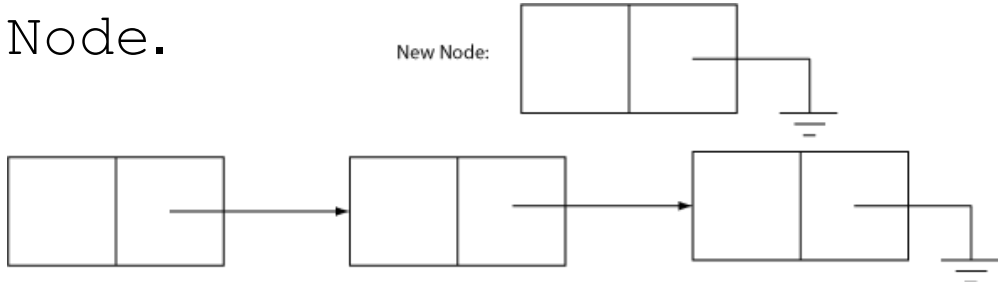
```
foundNode: Node := null
currentNode: Node := LinkedList.head
while currentNode != null && foundNode = null
    if currentNode.data = queryItem
        foundNode := currentNode
    currentNode := currentNode.next
```

The postcondition of this algorithm is that `foundNode` will be:

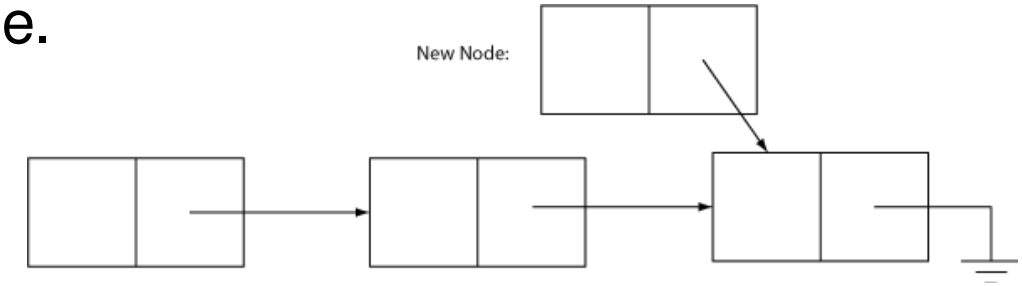
- The node containing the query item, or
- `null` if the query item is not in the list.

Inserting Elements into a Linked List

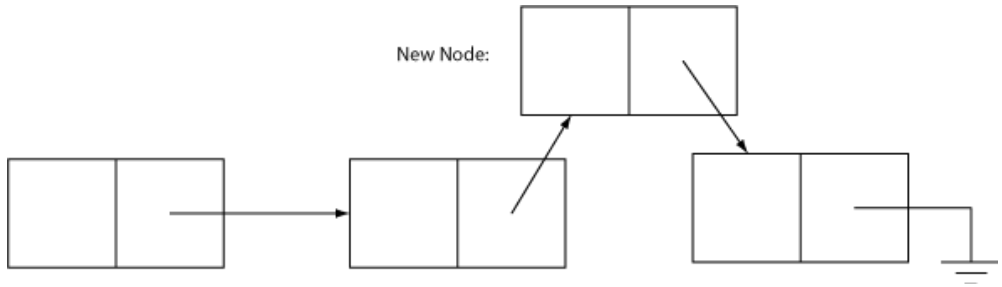
1. Find the existing `Node` to insert new element after.
2. Create a new `Node`.



3. Set the `next` reference of the new `Node` to the `next` reference of the existing node.



4. Set the `next` reference of the existing node to the new `Node`.



See LinkedList.java for the code.

Computing the Length of a Linked List

An algorithm for computing the length a linked list:

```
length: int := 0
currentNode: Node := LinkedList.head
while currentNode != null
    length := length + 1
    currentNode := currentNode.next
```

The postcondition of this algorithm is that `length` will be equal to the number of nodes in the list.

Generic Linked Lists

To make our LinkedList generic we only need to add a type parameter to the declaration:

```
public class GenericLinkedList<E> { ...
```

and replace `Object` with `E` in the body of the class.
See GenericLinkedList.java

Doubly Linked Lists

A doubly linked list simply adds a `previous` reference to the `Node` class so that the list can be traversed forwards or backwards.

```
    private class Node<E> {
        E data;
        Node<E> next;
        Node<E> previous;

        public Node(E data, Node<E> next, Node<E> previous) {
            this.data = data;
            this.next = next;
            this.previous = previous;
        }
    }
```

Doubly linked lists work the same, except that the algorithms for inserting and removing items requires a bit more link fiddling (have to set previous links as well).
See DoublyLinkedList.java.

Programming Exercises

Programming Exercise 1

- Add a `get(int index)` method to `GenericLinkedList`.
- Add a `remove(T item)` method to `GenericLinkedList`.

Programming Exercise 2

- Implement `public static int binarySearch(int[] a, int v)`. Return -1 if `v` is not in `a`.
- Bonus: implement `public static <T> int binarySearch(T[] a, T v, Comparator<? super T> c)`. Return -1 if `v` is not in `a`.
- Bonus: for either of the options above, implement your method using a recursive helper method.
- Bonus question: if we wanted to implement a similar method for a `Collection`, how would we do it? Could we define such a binary search method for any `Collection`?
- Bonus question 2: what is the running time (Big-O) of binary search?