

# CSE 114 – Computer Science I

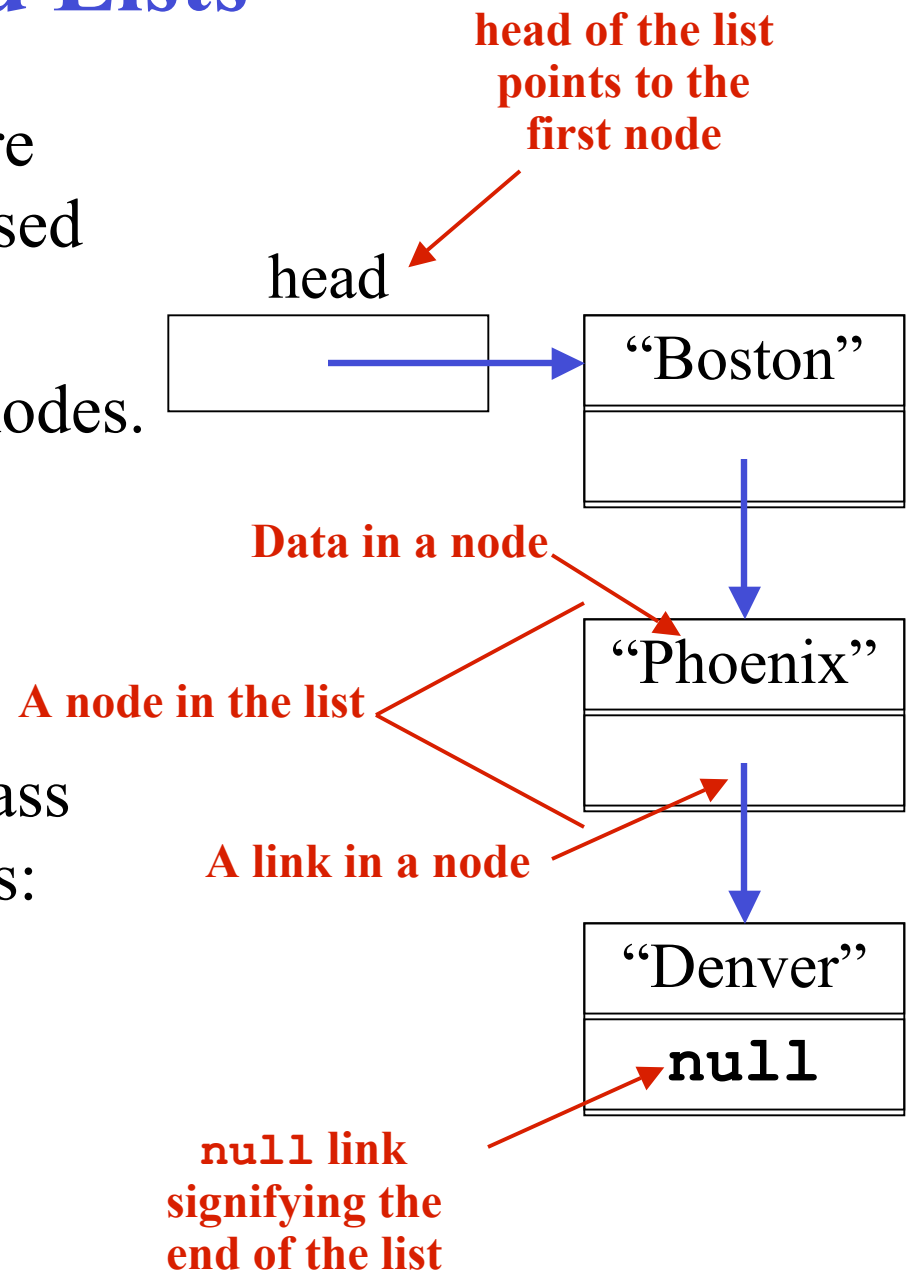
## Linked Lists



Route 66, Amarillo, TX

# Linked Lists

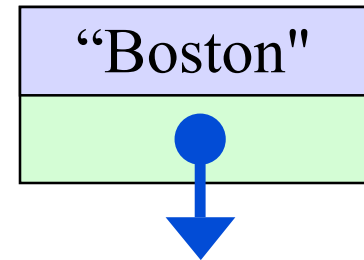
- A *Linked list* is a data structure (like arrays and **Vectors**), used to store data
- Consist of objects known as nodes.
- Each node has 2 components
  - a piece of data
  - a link to another node
- Each node is an object of a class that has two instance variables:
  - one for the data
  - one for the link



# Example: node class for a linked list of Strings

```
public class ListNode
{
    private String data;
    private ListNode next;

    public ListNode(String initData, ListNode initNext)
    {
        data = initData;
        next = initNext;
    }
}
```



- Two parameters for the constructor:
  - data value for the new node
  - next value for the new node
- If a node is the only element in the list, its next value should be **null**

# Don't forget accessor/mutator methods

- What are the method headers?
  - `public String getData()`
  - `public ListNode getNext()`
  - `public void setData(String changeData)`
  - `public void setNext(ListNode changeNext)`
- These are for **ListNode** objects, where each node stores **String** data.
- How about for a linked list of **PersonNodes**, where each node might store **Person** data?

# List-managing class

- Now we know how to make node classes
- We also want a class to manage the list
- A list manager should have the following:
  - The head of the list
    - Can access list elements through the head
  - Methods to manipulate the list
    - Add nodes to the list
    - Insert nodes into the list
    - Remove nodes from the list
    - Change data of a particular node in list
    - Sort nodes in a list
    - Search for a node in a list

# Node Inner Classes

```
public class LinkedListManager
{
    private ListNode head;
    <methods for LinkedListManager inserted here>

    private class ListNode
    {
        <Define ListNode instance variables>
        <Define ListNode methods>
    }
}
```

- Using an inner class makes **LinkedListManager** self-contained because it doesn't depend on a separate file.
- Making the inner class private makes it safer from the point of view of information hiding.

```
public class CityListManager
{
    private CityNode head;

    // METHODS TO MANIPULATE LIST WOULD BE ADDED HERE

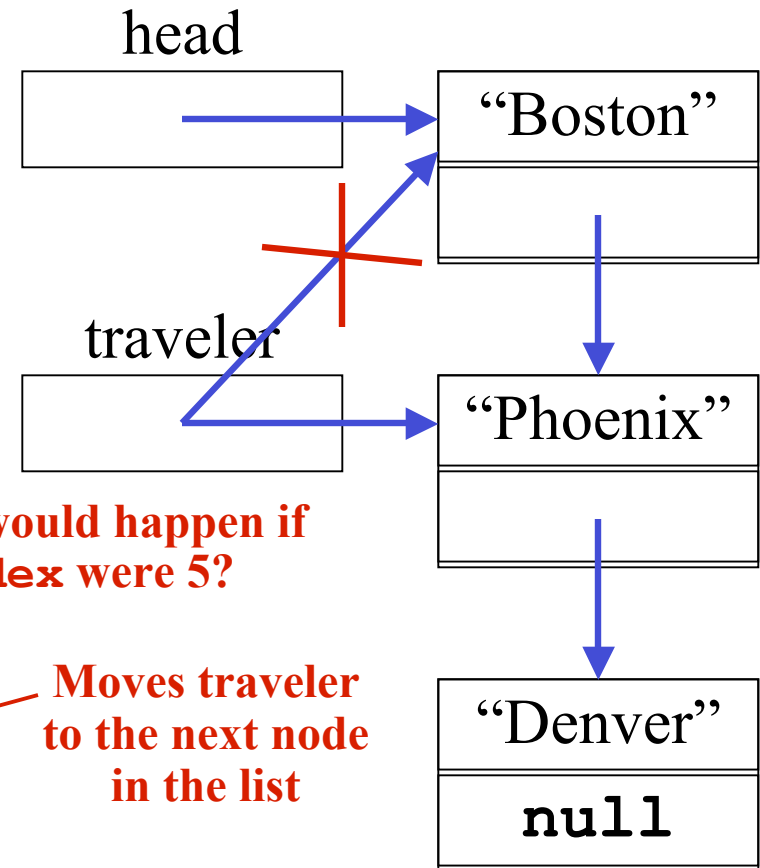
    private class CityNode
    {
        private String city;
        private CityNode next;

        public CityNode(String initCity, CityNode initNext)
        {
            city = initCity;
            next = initNext;
        }

        public String getCity() { return city; }
        public CityNode getNext() { return next; }
        public void setCity(String newCity) { city = newCity; }
        public void setNext(CityNode newNext) { next = newNext; }
    }
}
```

# Stepping Through a List

- In order to change a list (add, insert, remove, edit nodes)
  - Traverse through the list to the appropriate node
  - Change the necessary instance variable(s)



```
ListNode traveler;  
traveler = head;
```

Start at the beginning of the list

```
int index = 1;
```

What would happen if index were 5?

```
while ( (traveler != null)  
        && (index > 0) )  
{  
    traveler = traveler.getNext();  
    index--;  
}
```

Moves traveler to the next node in the list

```
System.out.println(traveler.getData());
```

**OUTPUT: Phoenix**

```
public class CityListManager
```

```
{
```

```
    private CityNode head;
```

```
    // METHODS GO HERE
```

```
    private class CityNode
```

```
    {
```

```
        private String city;
```

```
        private CityNode next;
```

```
        public CityNode(String initCity, CityNode initNext)
```

```
        {
```

```
            city = initCity;
```

```
            next = initNext;
```

```
        }
```

```
        public String getCity() { return city; }
```

```
        public CityNode getNext() { return next; }
```

```
        public void setCity(String newCity) { city = newCity; }
```

```
        public void setNext(CityNode newNext) { next = newNext; }
```

```
    }
```

**// Define 3 methods**

**1) Inserts a node at the beginning of the list**

**2) Removes the first node in the list**

**3) Removes the last node in the list**

## Method example: Inserting a Node

- To insert a node at the beginning of the list:

```
// INSIDE CityListManager CLASS
```

```
public void addANodeToStart(String addCity)
{
    head = new CityNode(addCity, head);
}
```

- The new node will point to the old start of the list, which is what **head** points to.
- The value of **head** is changed to point to the new node, which is now the first node in the list.
- What if **head** had previously pointed to a list with 20 nodes?

# Removing a Node

- To remove a node from the beginning of the list:

```
// INSIDE LIST MANAGER CLASS
public void deleteHeadNode ()
{
    if (head != null)
    {
        head = head.getNext ();
    }
}
```

- Doesn't try to delete from an empty list.
- Removes first element and sets head to point to the node that was second but is now first.
- What happens to the old **head**?

# Removing the last node in the list

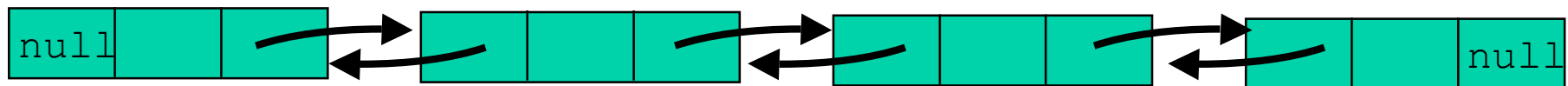
- To remove a node from the end of the list:

```
// INSIDE LIST MANAGER CLASS
public void removeLastNode()
{
    CityNode traveler = head;
    if (traveler != null)
    {
        if (traveler.getNext() == null)
            head = null;
        else
        {
            while (traveler.getNext().getNext() != null)
                traveler = traveler.getNext();
            traveler.setNext(null);
        }
    }
}
```

# A Doubly Linked List

- A doubly linked list allows the program to move backward as well as forward in the list.
- The beginning of the node class for a doubly-linked list would look something like this:

```
private class ListNode
{
    private Object data
    private ListNode next;
    private ListNode previous;
```



## Next Time

- More practice with linked lists
- Linked list variants: stacks and queues
- There may also be a quiz involving linked lists