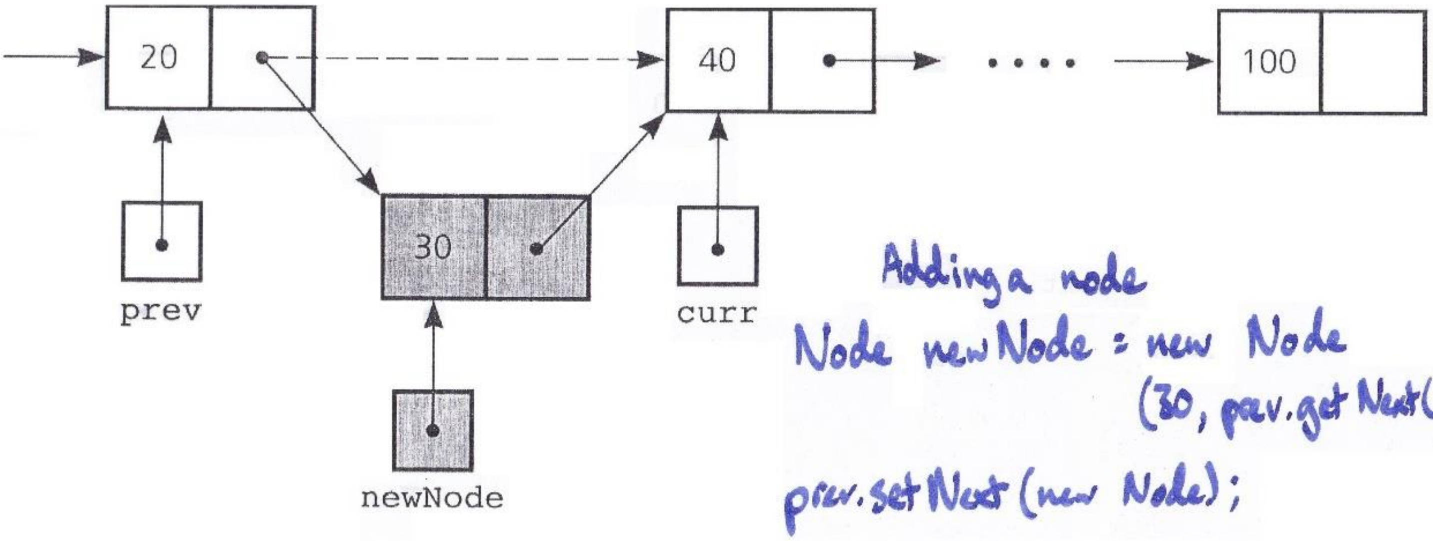
Inserting a new node into a sorted linked list



Adding a node

Node newNode = new Node
(30, prev.getNext());

prev.setNext (new Node);

Deleting a node from a linked list

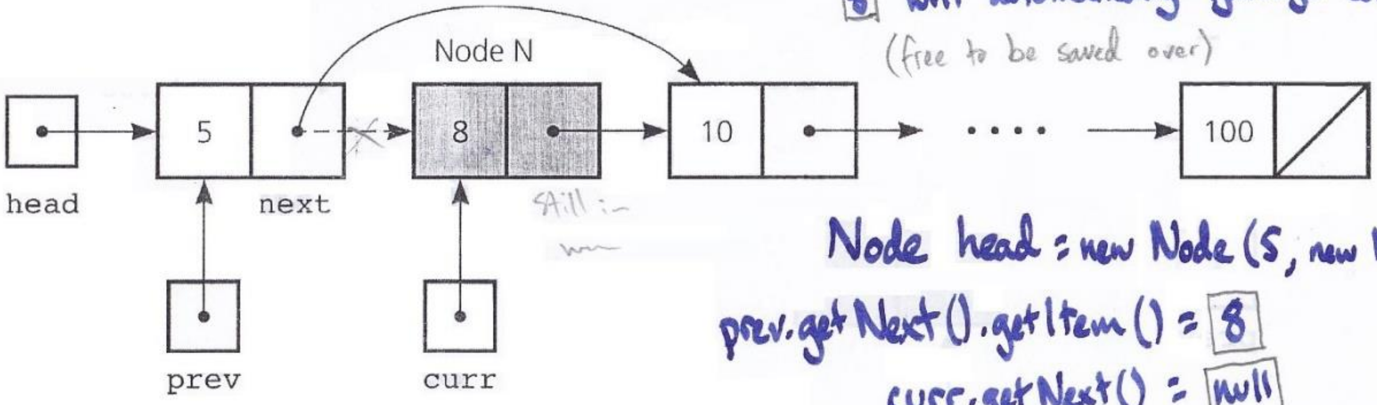
Deleting a node

prev.setNext (curr.getNext());

curr = null;

$\boxed{8}$ will automatically garbage collected

(free to be saved over)



Node head = new Node(5, new Node(8));

prev.getNext().getItem() = $\boxed{8}$

curr.getNext() = $\boxed{\text{null}}$

null pointer exception

prev.getNext() == curr ?

True

Arrays	Linked Lists
• ideal for applications with a constant size • very easy to find the *n*th element • hard to insert or delete into/from first slot of array • easy to insert or delete from the last slot • easy to apply our sorting algorithms to arrays are indexed	• ideal for applications with varying sizes • would have to search from beginning to end for *n*th element • easy to insert/delete the first item • hard to insert or delete to end since we have to find address of last item linked lists are not indexed

CSC 205 Linked List Notes (11/2/16)

The Wild & Powerful For Loop

```
for (prev = null,  curr = head;
     curr != null && newValue.compareTo(curr.getItem()) > 0;
     prev = curr, curr = curr.getNext() )
     {}
```

A Copy Constructor

- creates a unique copy of a dynamic data structure aka "deep copy"

```
public Object clone() throws CloneNotSupportedException
{
      Stack copy = new Stack();
      Node curr = top, prev = null;
                                      loop
      while (curr != null)
      {
              Node temp = new Node(curr.getItem());
              if (prev == null)
                      copy.top = temp;
              else
                      prev.setNext(temp);
              prev = temp;
              curr = curr.getNext();
      }

      return copy;
}
```

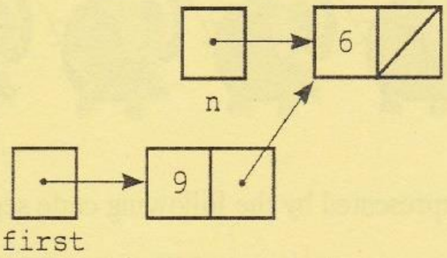
An Introduction To Linked Lists

/ How array lists are implemented

+ Bitsets + StringBuffer

+ BigInteger

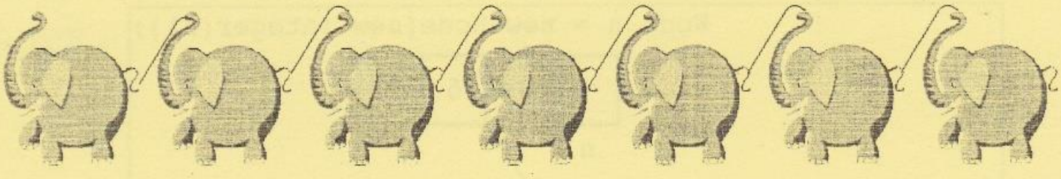
```
Node n = new Node(new Integer(6));
```



```
Node first = new Node(new Integer(9), n);
```

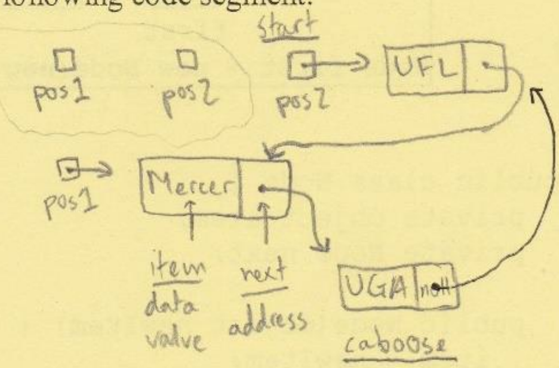
```
public class Node {
  private Object item;
  private Node next;

  public Node(Object newItem) {
    item = newItem;
    next = null; / "no address"
  } // end constructor

  public Node(Object newItem, Node nextNode) {
    item = newItem;
    next = nextNode;
  } // end constructor

  public void setItem(Object newItem) {
    item = newItem;
  } // end setItem

  public Object getItem() {
    return item;
  } // end getItem

  public void setNext(Node nextNode) {
    next = nextNode;
  } // end setNext

  public Node getNext() {
    return next;
  } // end getNext
} // end class Node
```

CSC 205 Linked List Worksheet



1. Draw the linked list that would be represented by the following code segment.

```
Node pos1 = null;
Node pos2 = null;

pos1 = new Node("Mercer");
pos1.setNext(new Node("UGA", null));
pos2 = new Node("UFL", pos1);
```



3) Circular linked list.

"Wrap around affect"

e.g. Music player

final node in linked list that doesn't link to next.

2. Using your linked list from Question #1, what would each of the following be equal to?

   a) `pos2.getNext().getItem()` Mercer

   b) `pos2.getNext().getNext().getItem()` UGA

   c) `pos1.getNext().getNext().getItem()` null.getItem() = Null Pointer Exception

   d) `(pos1) == (pos2.getNext())` Boolean. True

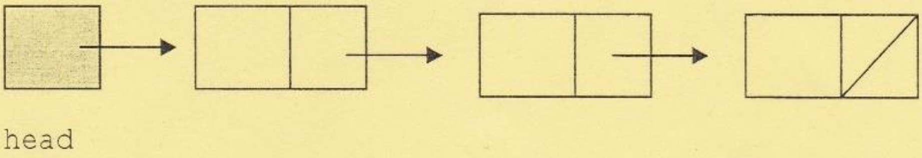
   e) `((Comparable)pos1.getItem()).compareTo((Comparable)pos2.getItem())`

      "Mercer".compareTo("UFL") = negative. ('M' - 'U')

3. Suppose you added the following line of code to the code segment that you used to create your linked list above. What would be the ramifications?

```
pos1.getNext().setNext(pos2);
```

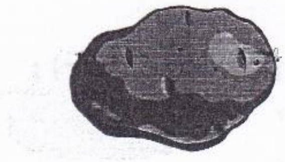
4. Using one line of code, change the value stored in the component of the last node in the list below to the integer 20. The first node in the list is pointed to by a head pointer.



head.getNext().getNext().setItem(20); third car

5. What would be the best data structure (array or dynamic linked list) to use for representing a list of the abbrevations of the 50 states? Why? Array. Linked lists are for applications that change rapidly

The Josephus Problem



The Josephus problem is the following game :

N people, numbered 1 to *N*, are sitting in a circle. Starting at person 1, a hot potato is passed. After *M* passes, the person holding the hot potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato. The last remaining person wins.

The Josephus problem arose in first century A.D., in a cave on a mountain in Israel, where Jewish zealots were besieged by Roman soldiers. The historian Josephus was among them. The zealots voted to form a suicide pact rather than surrender to the Romans. He suggested the game mentioned below. The hot potato was the sentence of death to the person next to the one who got the potato. Josephus rigged the game to get the last lot and convinced the intended victim that they should surrender. That is how we know about this game; in effect Josephus cheated!

What happens if you have 5 Players and 0 Passes ? ($N = 5$, $M = 0$)

1, 2, 3, 4, 5

What happens if you have 5 Players and 1 Pass? ($N = 5$, $M = 1$)

2, 4, 1, 5, 3

What data structure is best for implementing this problem? Why?

Circular linked so we can quickly relink the person behind us to the person in-front of us

What is the approximate "run-time" of this problem?

O(m.n)

```java
public class Josephus
{
  public static void main(String[] args)
  {
    int[] info = new int[2];
    Node trailer = null;   //  Used to traverse circle

    getInput (info);
    trailer = buildCircle (info[0], trailer);
    trailer = josephus (info[1], trailer);
  }

  public static void getInput(int[] info)
  {
    Scanner in = new Scanner(System.in);
    System.out.println("^^^ THE JOSEPHUS PROBLEM ^^^");
    System.out.print("Please Enter the Number of People (N) : ");
    info[0] = in.nextInt();
    System.out.print("Please Enter the Number of Passes (M) : ");
    info[1] = in.nextInt();
    System.out.println("^^^^^^^^^^^^^^^^^^^^^^^^^^^^");
  }

  public static Node buildCircle (int people, Node trailer)
  {
    Node temp = new Node(new Integer(1));
    Node ldata = temp;

    for (int i = 2; i <= people; i++)   {
      temp.setNext(new Node(new Integer(i)));
      temp = temp.getNext();
    }
    temp.setNext(ldata);  // Make list circular
    trailer  = temp;      // Record location of last person
    return trailer;
  }

  public static Node josephus (int passes, Node trailer)
  {
    Node potato;  // Assume potato starts at person 1
    System.out.print("\tOrder of Elimination :  ");

    while (trailer !=  trailer.getNext()){
       for (int i = 1; i<= passes; i++)   // Pass the hot potato
          trailer  = trailer.getNext();
       System.out.print(trailer.getNext().getItem() + " ");
       potato = trailer.getNext();
       trailer.setNext(potato.getNext());
    }
    System.out.println(trailer.getItem());
    return trailer;
  }
}
```