# CSC 1052 – Algorithms & Data Structures II:

# Linked Lists

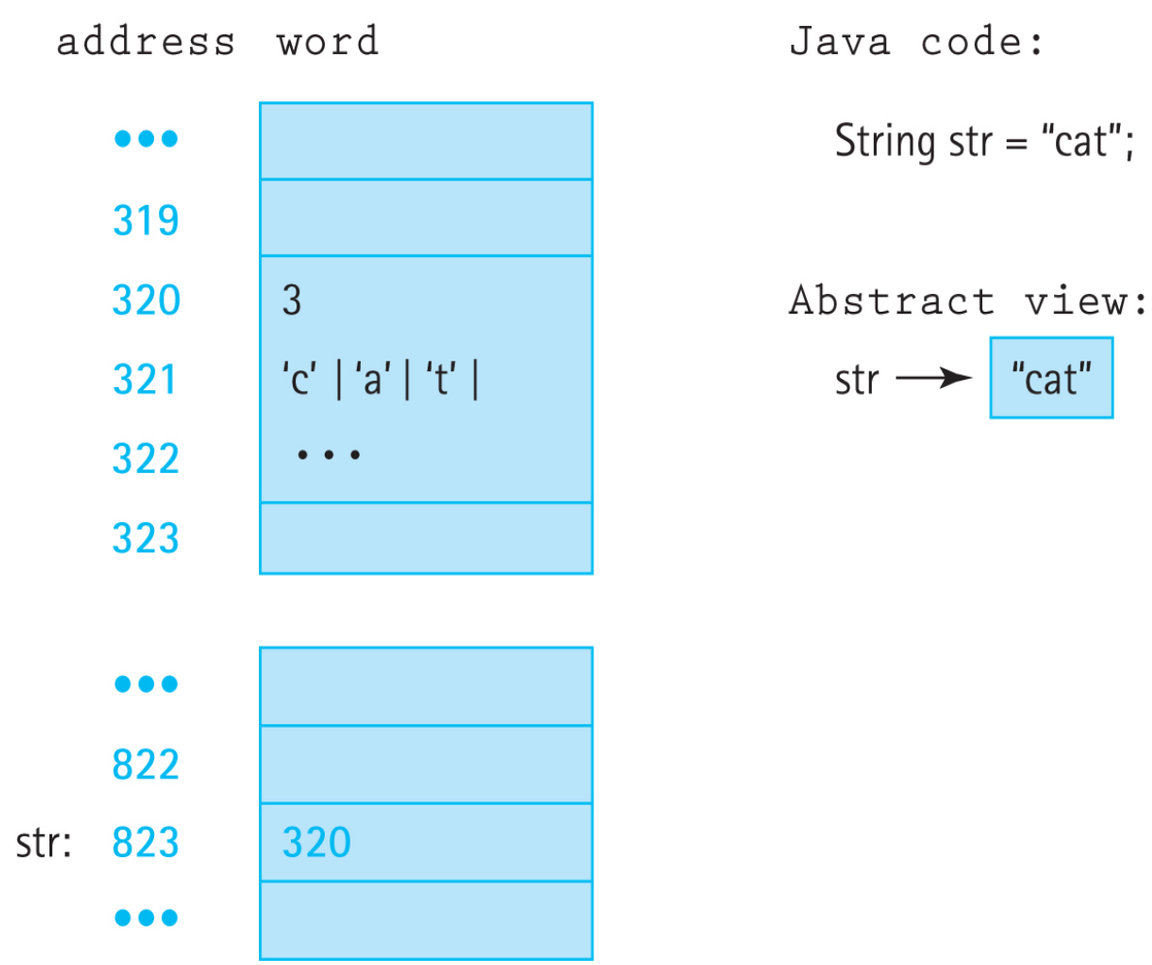
Professor Henry Carter
Spring 2017

# Recap

- Balanced expressions ensures delimiters are matched using a stack
- Postfix evaluation reduces math expression ambiguity at the cost of readability
- Remember:
  - Code
  - Test
  - Repeat

# Déjà Vu: Memory Revisited

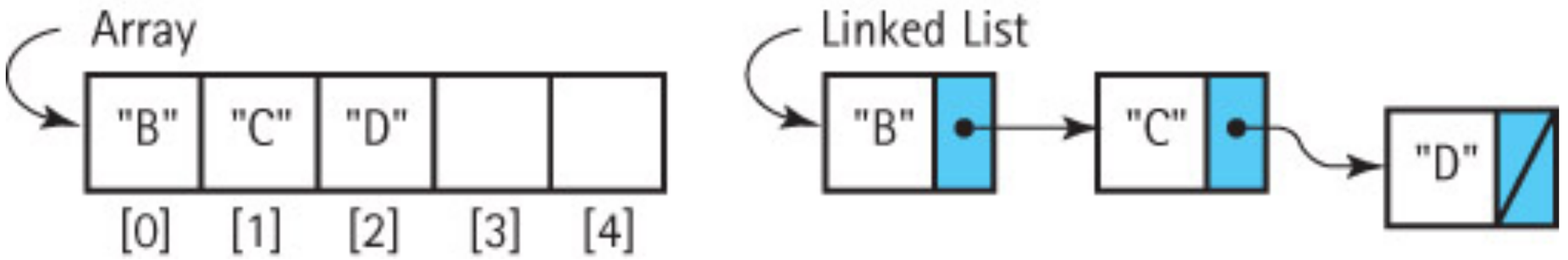
- Ordered list of memory words
- Direct vs indirect addressing
- Array storage



# Implementation-Dependent Data Structures

- Arrays
  - Contiguous memory allocated for each primitive value(or reference to an object)
- Linked Lists
  - A sequence of data values (or pointers) in random memory locations connected by pointers

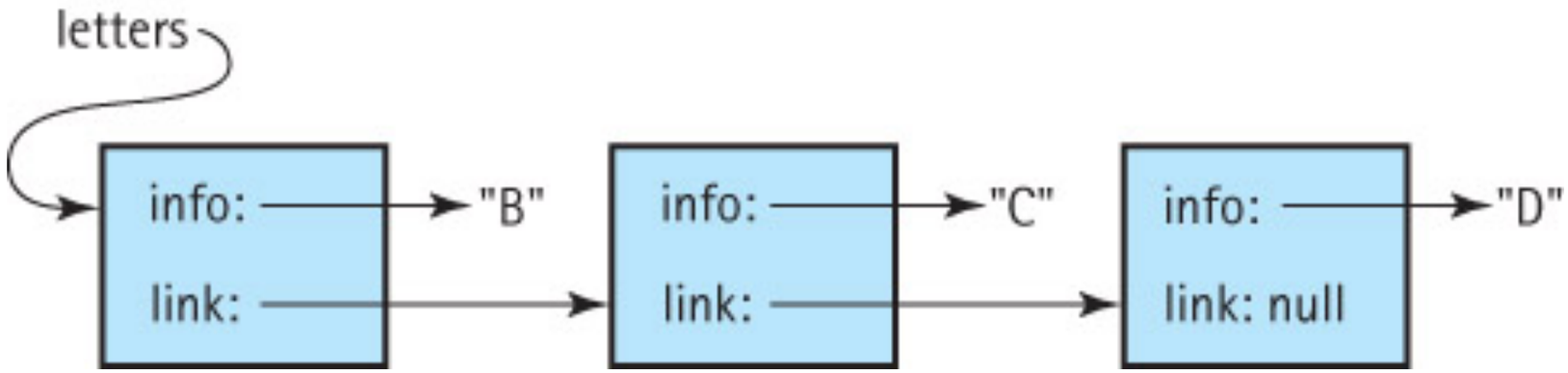
# A Visual



# Components

- Unlike arrays, where the next element is assumed to be next, the location of the next element has to be stored
- Elements in a linked list are encapsulated in a node
- Nodes contain two variables:
  - Data
  - Next Node

# A Visual: Nodes in Sequence



# What can they do?

- Add/remove elements
- Iterate through elements
- Access specified elements…?

# Recall: Orders of Growth

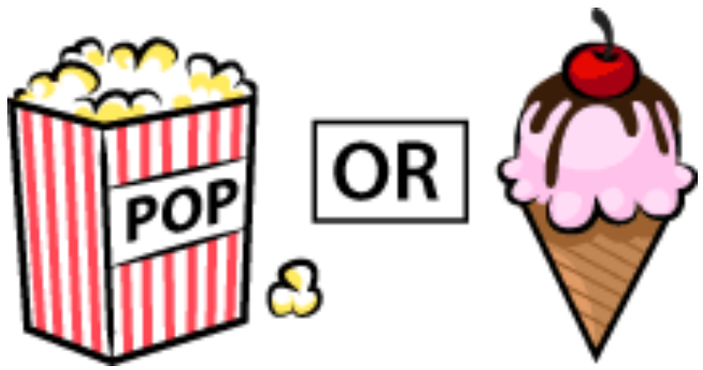
- We compare algorithms by approximating the operation counts
  - What operations are we counting with data structures?
- We group performance based on how fast the operation count grows
- Remember:
  - OOG for n + 9?
  - OOG for n + $n^2$?

# Exercise: Compare

- Determine the OOG for the following operations in both an array and a linked list
  - Accessing the first/last/middle element
  - Insertion at the front/end/middle
  - Deletion from the front/end/middle
  - Re-sizing (larger or smaller)

# Tradeoffs

- Linked lists are more memory efficient
- Arrays are more time efficient
  - Except for which operation?



# Advantageous Scenarios

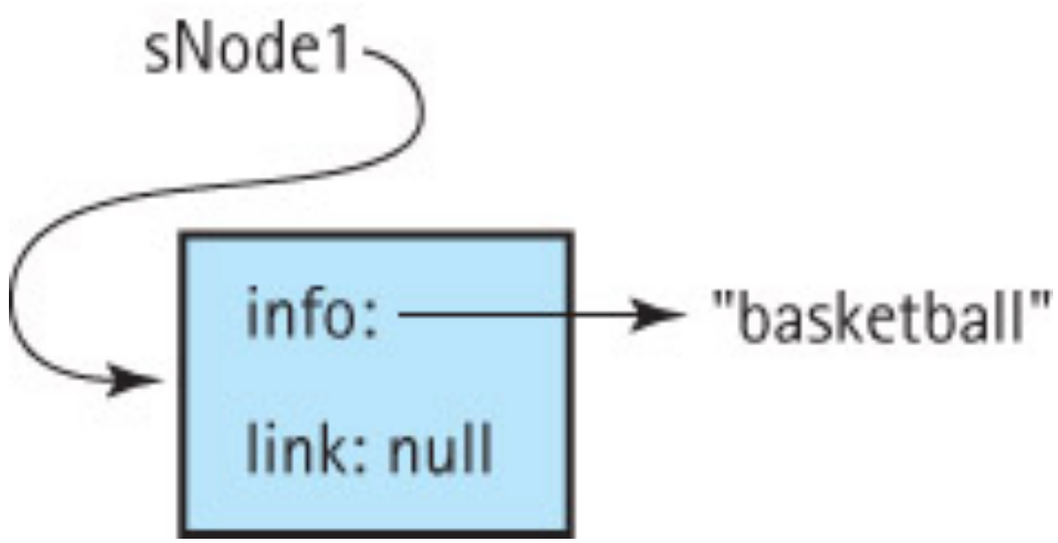
- Absolutely minimize allocated space
- Data structures that do not allow random access
  - (can you think of one?)
- Data structures that require specific ordering
  - Shifting values in an array is costly

# Linked List Implementation

- Stored value: the head
- Intermediate values: nodes
- How do we define and allocate nodes?

# LLNode Class

- Class variables
- Constructor
- Getters/Setters



# LLNode Source

```
//-------------------------------------------------
// LLNode.java    by Dale/Joyce/Weems      Chapter 2
//
// Implements <T>  nodes for a Linked List.
//-------------------------------------------------

package support;

public class LLNode<T>
{
   protected T info;
   protected LLNode<T> link;

   public LLNode(T info)
   {
      this.info = info;
      link = null;
   }

   public void setInfo(T info){ this.info = info;}

   public T getInfo(){ return info; }

   public void setLink(LLNode<T> link){ this.link = link;}

   public LLNode<T> getLink(){ return link;}
}
```

# Traversal Pseudocode

# Traversal Java Code

```
LLNode<String> currNode = letters;
while (currNode != null)
{
  System.out.println(currNode.getInfo());
  currNode = currNode.getLink();
}
```

# Front Insertion Pseudocode

# Front Insertion Java Code

```
LLNode newNode = new LLNode(data);
newNode.setLink(letters);
letters = newNode;
```

# Implementation Exercises

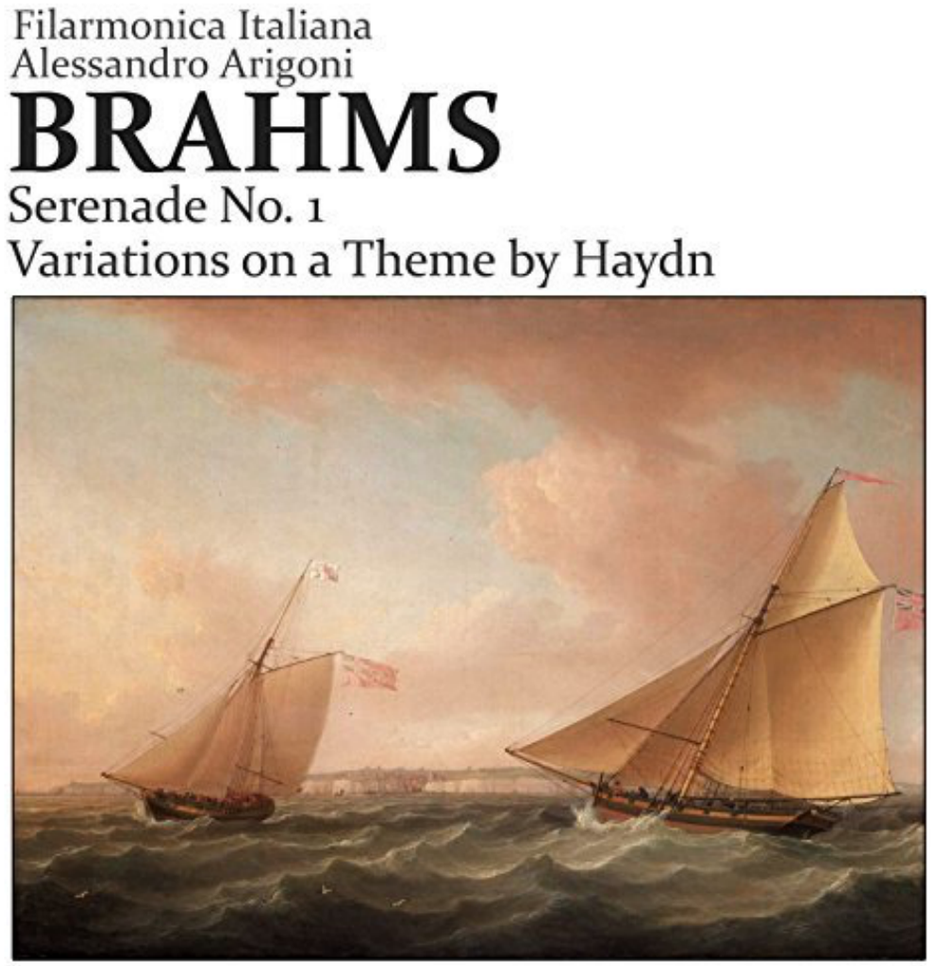
- Go to www.hackerrank.com
- Navigate to the Data Structures -> Linked Lists exercises
- Complete the three insert exercises:
  - Tail
  - Head
  - Specified location

# Notable Restrictions

- Where can you iterate?
- Where can you start?
- What additional information might you wish to store?

# Linked List Variations

- Circular linked list
- Tail pointer
- Doubly linked list
- Size variable



# Recap

- Linked list allow data to be stored in random locations in memory
- Typically require more time to access in exchange for more efficient memory usage
- Implementation pitfalls frequently related to pointers
- Variants allow for more flexible access at the cost of method complexity

# Next Time...

- Dale, Joyce, Weems Chapter 2.8, 2.10
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/