LINKED LISTS

MSCI 240: Algorithms & Data Structures

lecture summary

traversing a linked data structure

`LinkedIntList` class

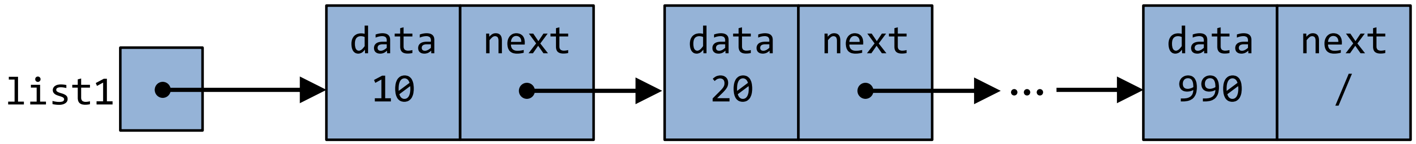
doubly-linked list

linked lists vs. arrays

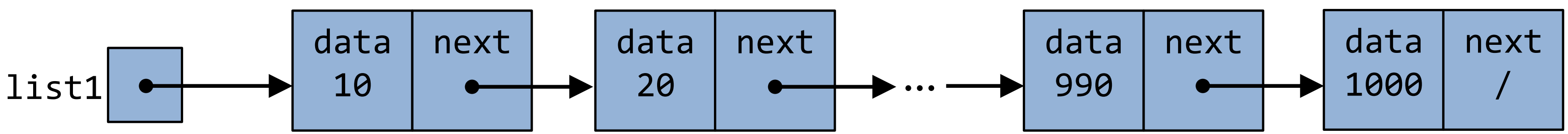
<table>
<tr><th>Topic</th><th>Building Java Programs</th><th>Algorithms (Sedgewick)</th></tr>
<tr><td>classes, ADTs</td><td>chapter 8</td><td>1.2</td></tr>
<tr><td>arrays</td><td>chapter 7</td><td></td></tr>
<tr><td>ArrayList<T></td><td>chapter 10</td><td>1.3</td></tr>
<tr><td>Stack/Queue</td><td>chapter 14, (11)</td><td>1.3</td></tr>
<tr><td>LinkedList</td><td>chapter 16</td><td>1.3</td></tr>
<tr><td>Complexity</td><td rowspan="3">chapter 13</td><td>1.4</td></tr>
<tr><td>Searching</td><td>pp. 46-47</td></tr>
<tr><td>Sorting</td><td>chapter 2.1-2.3</td></tr>
<tr><td>Recursion</td><td>chapter 12</td><td>1.1 (p. 25)</td></tr>
<tr><td>BSTs</td><td>chapter 17</td><td>chapter 3.1-3.2</td></tr>
<tr><td>Dictionaries</td><td>chapter 18.1</td><td>chapter 3.4</td></tr>
<tr><td>Graphs</td><td>N/A (Wikipedia good)</td><td>chapter 4.1</td></tr>
<tr><td>Heaps/Priority Queues</td><td>chapter 18.2</td><td>chapter 2.4</td></tr>
</table>

traversing a linked data structure

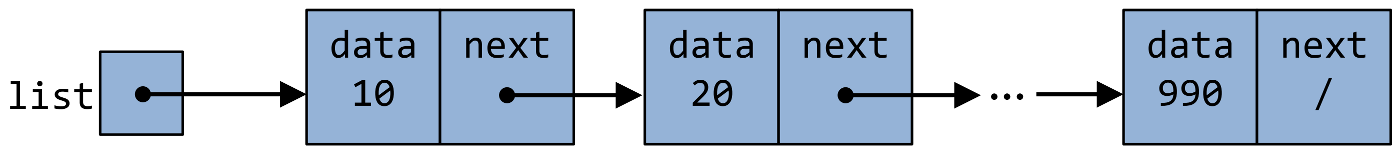
what set of statements turns this picture:



into this?



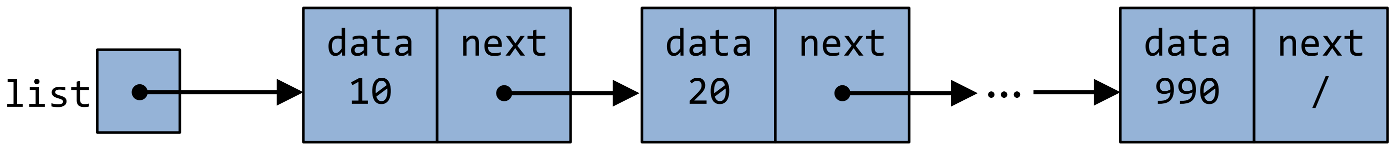
suppose we have a long chain of list nodes:



we don't know exactly how long the chain is

how would we print the data values in all the nodes?

algorithm pseudocode



start at the front of the list

while (there are more nodes to print):

print the current node's data

go to the next node

how do we walk through the nodes of the list?

```
list = list.next;    // is this a good idea?
```

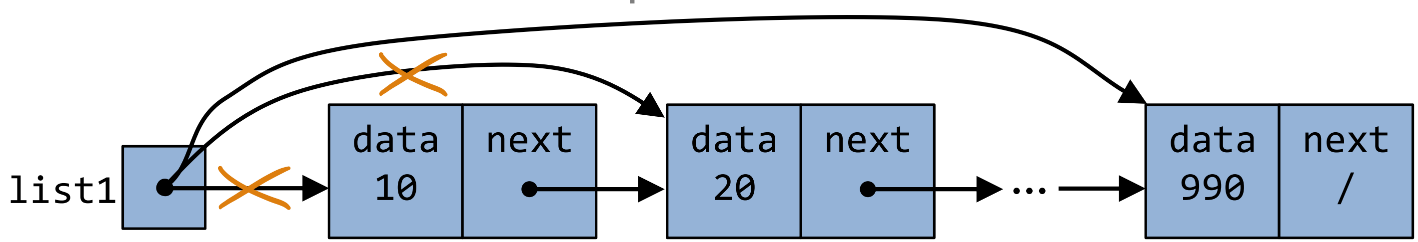
traversing a linked data structure?

one (bad) way to print every value in the list:

```
while (list != null) {
    System.out.println(list.data);
    list = list.next;    // move to next node
}
```

what's wrong with this approach?

(It loses the linked list as it prints it!)



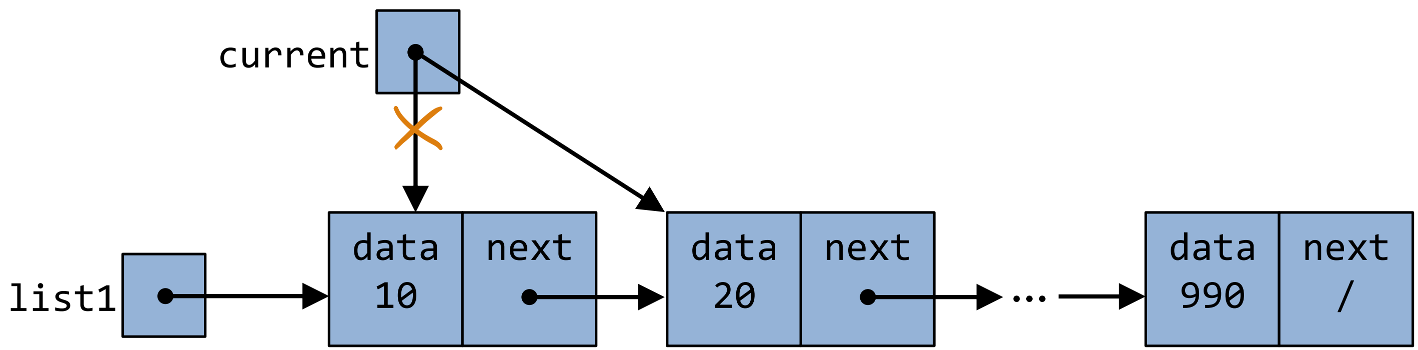
don't change `list`—make another variable, and change that

a `ListNode` variable is not a `ListNode` object (it's a reference)

```
ListNode current = list;
```

what happens to the picture when we write:

```
current = current.next;
```

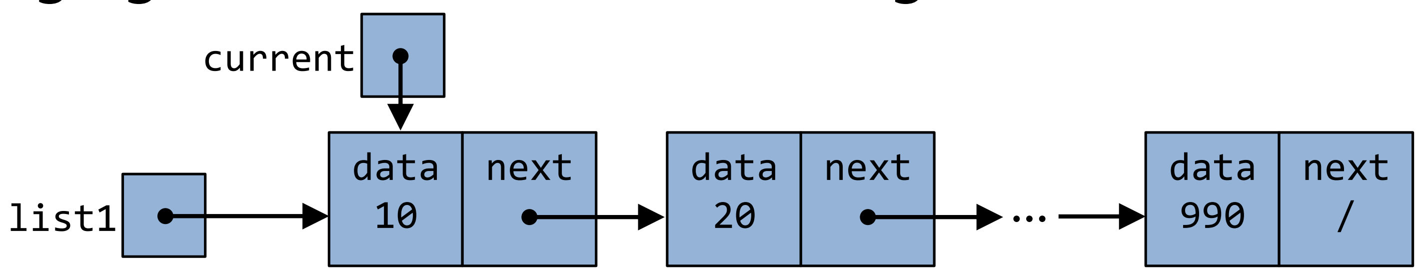


traversing a linked data structure correctly

the correct way to print every value in the list

```
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;  // move to next node
}
```

changing `current` does not damage the list



linked list vs. array

algorithm to print list values:

```
ListNode front = ...;

ListNode current = front;
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```

initialization

test

update

similar to array code:

```
int[] a = ...;

int i = 0;
while (i < a.length) {
    System.out.println(a[i]);
    i++;
}
```

traversing linked data structure summary

to traverse a linked data structure, you need to keep track of the current element using a second node reference (`current`)

the loop has the same parts as a for loop:

initialization:
```
ListNode current = front;
```
test:
```
while (current != null)
```
update:
```
current = current.next;
```

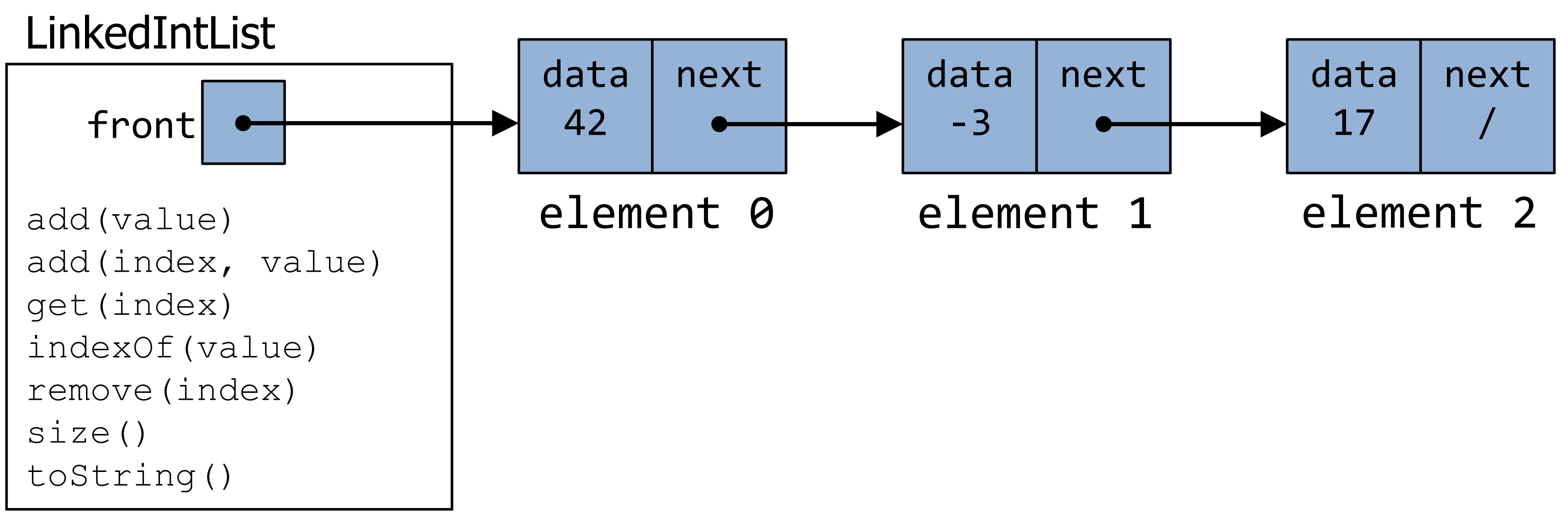
`LinkedIntList` class

let's write a collection class named `LinkedIntList`

- has the same methods as `ArrayListOfDouble`:
- **`add`**, **`add`**, **`get`**, **`indexOf`**, **`remove`**, **`size`**, **`toString`**

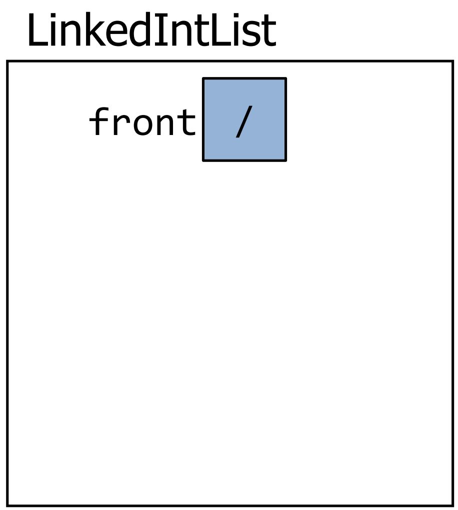
the list is internally implemented as a chain of linked nodes

- the `LinkedIntList` keeps a reference to its `front` as a field
- **`null`** is the end of the list; a **`null`** `front` signifies an empty list



`LinkedIntList` class version 1

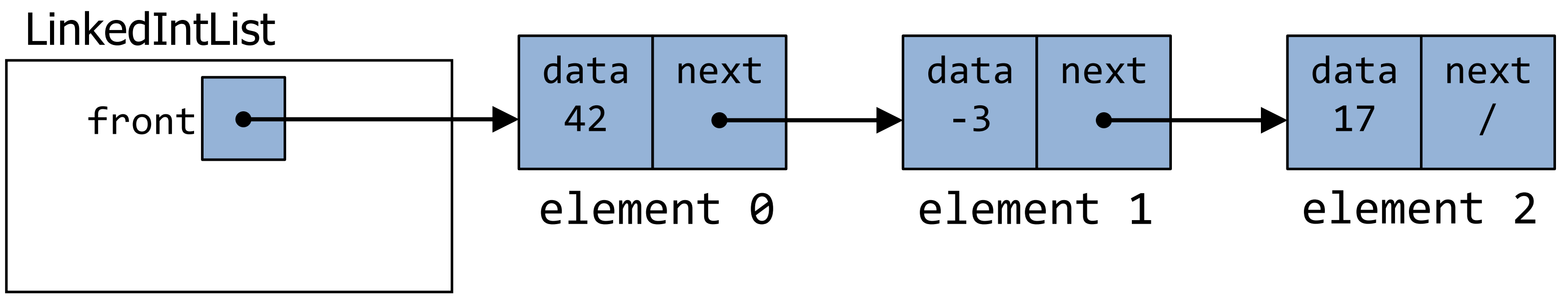
```java
public class LinkedIntList {
    private ListNode front;

    public LinkedIntList() {
        front = null;
    }

    // methods go here
}
```



the `add` method

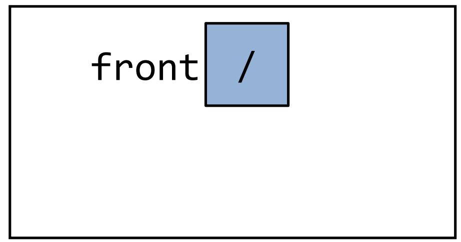
how do we add a new node to the end of a list?

does it matter what the list's contents are before the add?

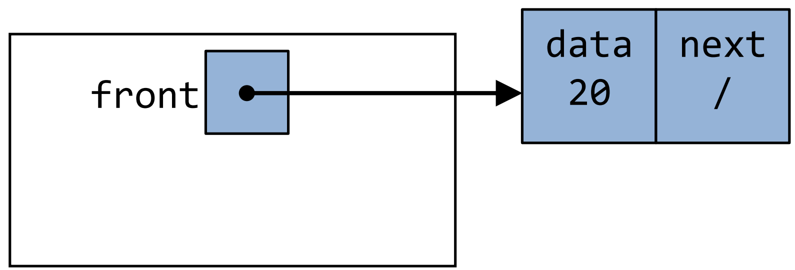


adding to an empty list

before adding 20:



after:



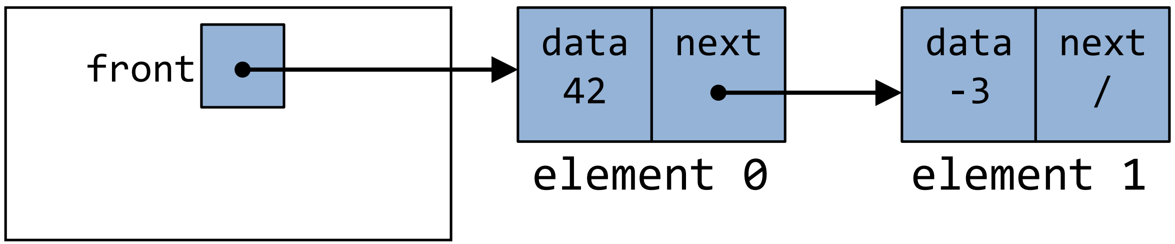
we must create a new node and attach it to the list

the `add` method, 1st try

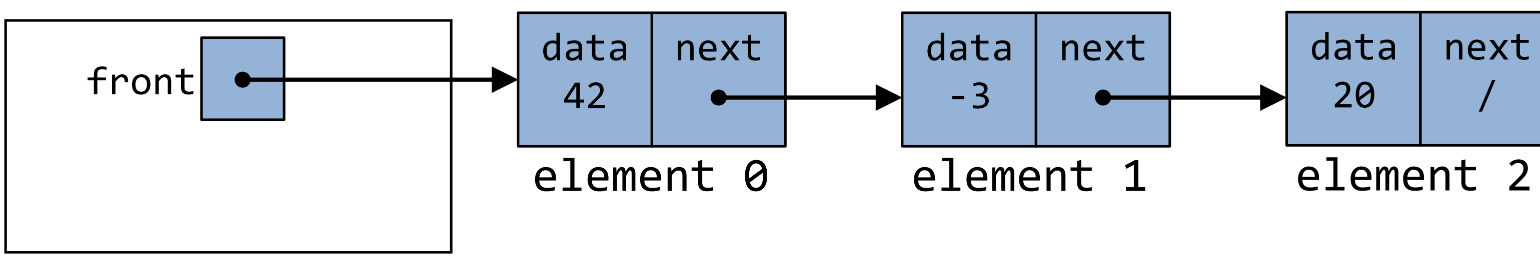
```java
// Adds the given value to the end of the list
public void add(int value) {
    if (front == null) {
        // adding to an empty list
        front = new ListNode(value);
    } else {
        // adding to the end of an existing list

        // ...

    }
}
```

adding to an non-empty list

before adding 20 to end of the list:

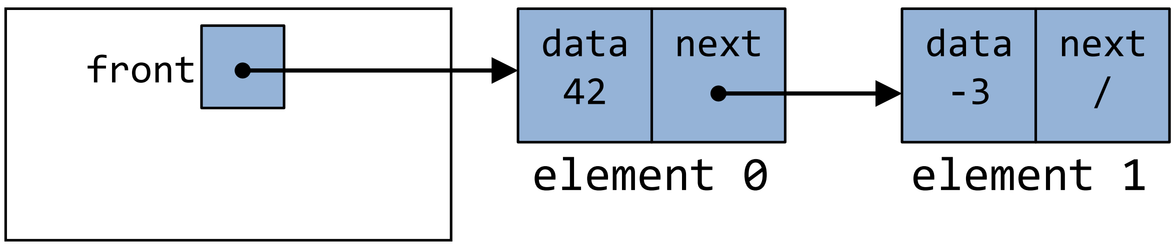


after:



don't fall off the edge!

to add/remove from a list, you must modify the `next` reference of the node before the place you want to change.



where should `current` be pointing, to add 20 at the end?

what loop test will stop us at this place in the list?

```java
// Adds the given value to the end of the list
public void add(int value) {
    if (front == null) {
        // adding to an empty list
        front = new ListNode(value);
    }
    else {
        // adding to the end of an existing list
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new ListNode(value);
    }
}
```

the get method

```java
// Returns value in list at given index.
// Precondition: 0 <= index < size()
public int get(int index) {
    ListNode current = front;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    return current.data;
}
```

conceptual questions

what's the difference between a `LinkedIntList` and a `ListNode`?

answer:

- a list consists of `0` to many node objects
- each node holds a single data element value

conceptual questions

what's the difference between an empty list and a **`null`** list?

how do you create each one?

`null` list:

```
LinkedIntList list = null;
```

empty list:

```
LinkedIntList list = new LinkedIntList();
```

conceptual questions

why are the fields of `ListNode` public? is this bad style?

answer:

it's okay that the node fields are public, because client code never directly interacts with `ListNode` objects

conceptual questions

what effect does this code have on a `LinkedIntList`?

```
ListNode current = front;
current = null;
```

answer: the code doesn't change the list, you can change a list only in one of the following two ways:

- modify its `front` field value
- modify the `next` reference of a node in the list

`LinkedIntList` summary

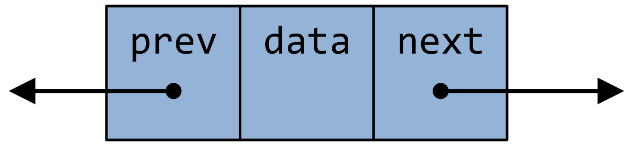
a linked list is another collection that abstracts the adding, removing, getting, etc. of a linked data structure

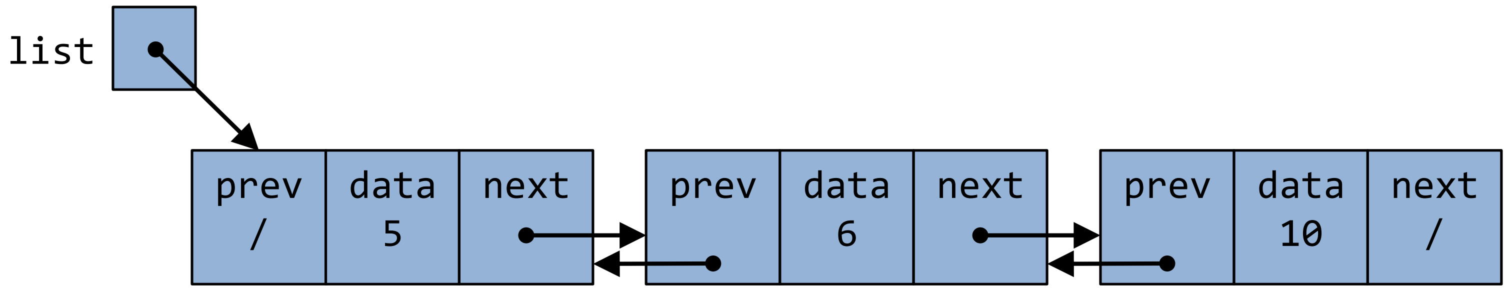
each method often needs to test for special cases:

- is the list empty?
- is something being done to the front / middle / end of the list?
- be careful not to "fall off the edge" (or go to far)

doubly-linked list

```java
public class Node<E> {
    public E data;
    public Node<E> next;
    public Node<E> prev;

    public Node(E data) {
        this.data = data;
    }

    public Node(E data, Node<E> prev) {
        this.data = data;
        this.prev = prev;
    }

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





exercise: write code to create this structure using only one `Node<E>` variable, called `list`

`LinkedList` vs. array

LinkedList
~~arrays~~:

separated
~~contiguous~~ storage

~~compact space requirements~~

to specific elements
~~random~~ access in “constant time”

variable
~~fixed~~ length

how much space does:

- an array of n integers use?
- a singly-linked list of n integers use?
- a doubly-linked list of n integers use?

data locality

- how close together are elements in an array?
- how close together are nodes in a linked list?

clicker questions

the pace of the class so far is:

A. way too slow
B. slightly too slow
C. just right
D. slightly too fast
E. way too fast

when used to implement a list, __________ can run out of space and require considerable work to allocate more space

A. arrays
B. linked lists
C. there is no difference between arrays and linked lists when it comes to running out of space

storage of n elements ($n > 0, n \in \mathbb{Z}$) would require less space in this data structure:

A. array
B. linked list
C. both an array and a linked list will require the same amount of space to hold n elements

you have an algorithm that requires a set of elements to be stored in a list that provides constant time, random access to the elements; what data structure do you use to store the elements?

A. array
B. linked list
C. Java class
D. text file
E. none of the above provide constant time, random access to a list of elements

modern CPUs run faster on algorithms that have good data locality (algorithms access memory in a pattern such that each access is followed by another close to it in RAM); which data structure will likely result in poor data locality?

A. array
B. linked list
C. both will cause any algorithm to likely have poor data locality
D. neither will cause any algorithm to likely have poor data locality

stacks are more efficient than queues

A. true

B. false

C. this is a stupid question, efficient at what and in what terms?

next class:

performance