

LEC 04

CSE 123

Linked Nodes

Questions during Class?
Raise hand or send here

sli.do #cse123



BEFORE WE START

Talk to your neighbors:

What is your favorite type of cookie?

Respond on sli.do!

Instructor: Nathan Brunelle

TAs:

Cora	Chris	Crystal	Anya	Evan
Jonah	Nhan	Saachi	Ishita	Sahana
Chloe	Shiven	Nate	Renee	Yuntong
Amiya	Shashmit	Akshaya	Vihan	Shreedhaarini
Roshan	Adiyan	Nancy	Rachel	Ananya
Ryuta	Mimi	Isaac	Megan	

Lecture Outline: Announcements

- **Announcements** 
- Reference Semantics Review
- Contiguous / Non-Contiguous Memory Review
- ListNode Practice

Announcements

- Programming Project 0 due one week from today (Oct. 21)
- Computer Quiz 0 starts one week from tomorrow (Oct. 22) and goes for one week (Oct. 29)
 - See Ed Announcement about policies/logistics
 - Bring your own laptop (and charge it!)
 - You may have a 1-page (front and back) notes sheet
 - You may only use 2 browser tabs during the quiz: the Ed lesson for CQ0 and for P0
 - The code quality guide and commenting guide will be embedded in the quiz
 - Sign ups begin tomorrow (in Canvas)
 - Practice quiz posted tomorrow
 - Builds on P0

Lecture Outline: Reference Semantics Review

- Announcements
- **Reference Semantics Review** ◀
- Contiguous / Non-Contiguous Memory Review
- ListNode Practice

Reference Semantics (1)

- In Java, variables are treated two different ways:

Value Semantics	Reference Semantics
Primitive types (int, double, boolean) + Strings*	Object types (int[], Scanner, ArrayList)
Values stored locally	Values stored in memory, reference stored locally
Initialization copies value (many copies of value)	Initialization copies reference (only one value)

<pre>int x = 10; int y = x; y++; // x remains unchanged</pre>	<pre>int[] x = new int[5]; int[] y = x; y[0]++; // x[0] changed</pre>
---	--

We often draw “**reference diagrams**” to keep track of everything

*Kind of



Reference Semantics (2)

- In Java, variables are treated two different ways:

Value Semantics	Reference Semantics
Primitive types (int, double, boolean) + Strings*	Object types (int[], Scanner, ArrayList)
Values stored locally	Values stored in memory, reference stored locally
Initialization copies value (many copies of value)	Initialization copies reference (only one value)

<pre>int x = 10; int y = x; y++; // x remains unchanged</pre>	<pre>int[] x = new int[5]; int[] y = x; y[0]++; // x[0] changed</pre>
---	--

We often draw “**reference diagrams**” to keep track of everything

*Kind of



Reference Semantics (3)

- In Java, variables are treated two different ways:

Value Semantics	Reference Semantics
Primitive types (int, double, boolean) + Strings*	Object types (int[], Scanner, ArrayList)
Values stored locally	Values stored in memory, reference stored locally
Initialization copies value (many copies of value)	Initialization copies reference (only one value)

<pre>int x = 10; int y = x; y++; // x remains unchanged</pre>	<pre>int[] x = new int[5]; int[] y = x; y[0]++; // x[0] changed</pre>
---	--

We often draw “**reference diagrams**” to keep track of everything

*Kind of



Reference Semantics (4)

- In Java, variables are treated two different ways:

Value Semantics	Reference Semantics
Primitive types (int, double, boolean) + Strings*	Object types (int[], Scanner, ArrayList)
Values stored locally	Values stored in memory, reference stored locally
Initialization copies value (many copies of value)	Initialization copies reference (only one value)

<pre>int x = 10; int y = x; y++; // x remains unchanged</pre>	<pre>int[] x = new int[5]; int[] y = x; y[0]++; // x[0] changed</pre>
---	--

We often draw “**reference diagrams**” to keep track of everything



Reference Semantics (5)

- In Java, variables are treated two different ways:

Value Semantics	Reference Semantics
Primitive types (int, double, boolean) + Strings*	Object types (int[], Scanner, ArrayList)
Values stored locally	Values stored in memory, reference stored locally
Initialization copies value (many copies of value)	Initialization copies reference (only one value)

<pre>int x = 10; int y = x; y++; // x remains unchanged</pre>	<pre>int[] x = new int[5]; int[] y = x; y[0]++; // x[0] changed</pre>
---	--

We often draw “**reference diagrams**” to keep track of everything



Reference Semantics (6)

- In Java, variables are treated two different ways:

Value Semantics	Reference Semantics
Primitive types (int, double, boolean) + Strings*	Object types (int[], Scanner, ArrayList)
Values stored locally	Values stored in memory, reference stored locally
Initialization copies value (many copies of value)	Initialization copies reference (only one value)

<pre>int x = 10; int y = x; y++; // x remains unchanged</pre>	<pre>int[] x = new int[5]; int[] y = x; y[0]++; // x[0] changed</pre>
---	--

We often draw “**reference diagrams**” to keep track of everything



Lecture Outline: Contiguous / Non-Contiguous Memory Review

- Announcements
- Reference Semantics Review
- **Contiguous / Non-Contiguous Memory Review** ◀
- ListNode Practice

Contiguous vs. Non-contiguous: Memory

- Computer memory = one really, *really* big array.

Memory

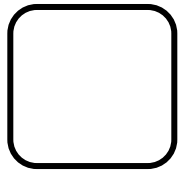
85	47	-51	44	-38	35	-58	79	27	-14
-24	-38	-66	-27	36	-1	23	20	31	-40
-34	38	37	-52	-15	99	6	68	-67	-58
13	-17	-85	-99	-20	-33	54	38	-66	8
36	24	27	90	-32	72	-73	11	-85	29
-90	-64	29	-27	91	64	28	-97	44	59
-68	76	-1	-6	-52	77	21	37	80	69

Contiguous vs. Non-contiguous: array (1)

- Computer memory = one really, *really* big array.
 - `int[] arr = new int[7];`

Memory

arr

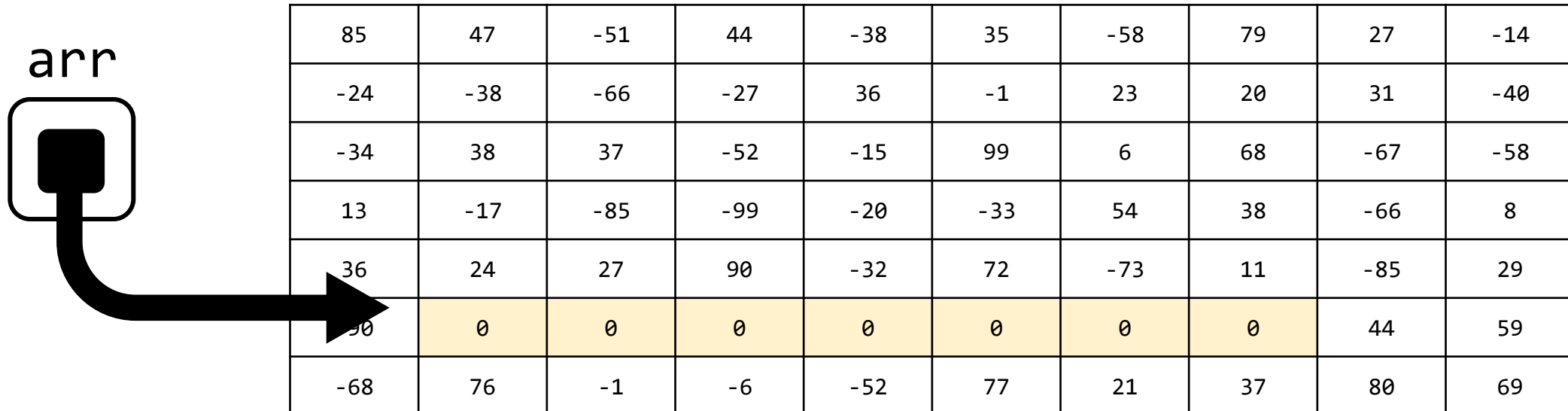


85	47	-51	44	-38	35	-58	79	27	-14
-24	-38	-66	-27	36	-1	23	20	31	-40
-34	38	37	-52	-15	99	6	68	-67	-58
13	-17	-85	-99	-20	-33	54	38	-66	8
36	24	27	90	-32	72	-73	11	-85	29
-90	-64	29	-27	91	64	28	-97	44	59
-68	76	-1	-6	-52	77	21	37	80	69

Contiguous vs. Non-contiguous: array (2)

- Computer memory = one really, *really* big array.
 - `int[] arr = new int[7];`

Memory



85	47	-51	44	-38	35	-58	79	27	-14
-24	-38	-66	-27	36	-1	23	20	31	-40
-34	38	37	-52	-15	99	6	68	-67	-58
13	-17	-85	-99	-20	-33	54	38	-66	8
36	24	27	90	-32	72	-73	11	-85	29
50	0	0	0	0	0	0	0	44	59
-68	76	-1	-6	-52	77	21	37	80	69

We call this “contiguous” memory

ListNode

- Java class representing a “**node**”
- Two fields to store discussed state:
 - Fields are public?! We’ll come back to this
- Why can `ListNode` be a field in the `ListNode` class?

```
public class ListNode {  
    public int data;  
    public ListNode next;  
}
```

Contiguous vs. Non-contiguous: ListNode (1)

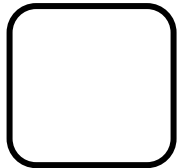
- Computer memory = one really, *really* big array.

```
ListNode list = new ListNode(1);
```

```
list.next = new ListNode(2);
```

Memory

list



85	47	-51	44	-38	35	-58	79	27	-14
-24	-38	-1	-27	36	-1	23	20	31	-40
-34	38	37	-52	-15	99	6	68	-67	-58
13	-17	-85	-99	-20	-33	54	38	-66	8
36	24	27	90	-32	72	-73	11	-85	29
-90	-64	29	-27	91	64	28	-97	44	59
-68	76	-1	-6	-52	77	21	37	80	69

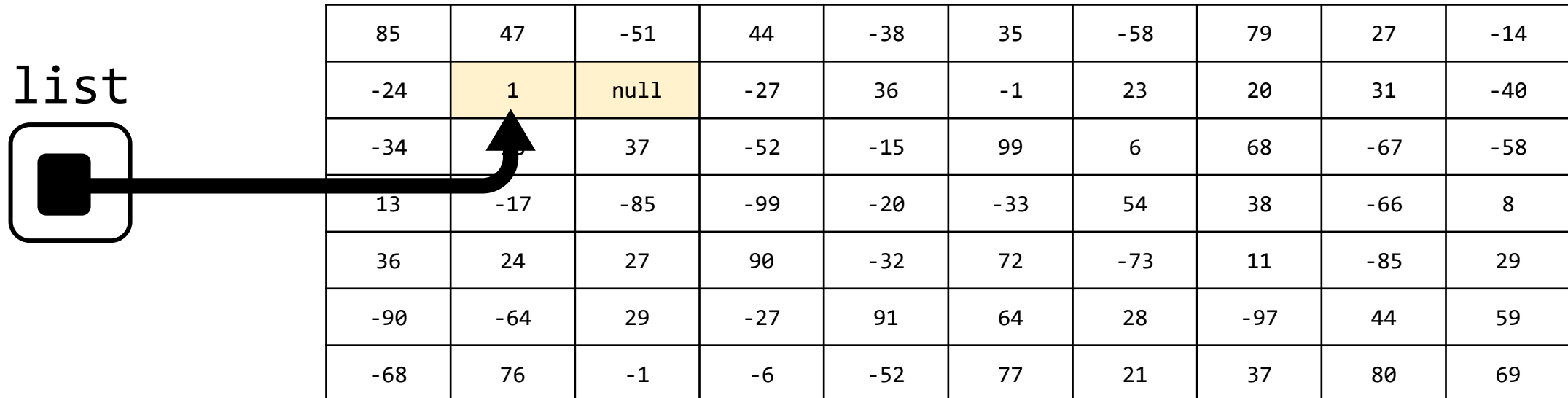
Contiguous vs. Non-contiguous: ListNode (2)

- Computer memory = one really, *really* big array.

```
ListNode list = new ListNode(1);
```

```
list.next = new ListNode(2);
```

Memory



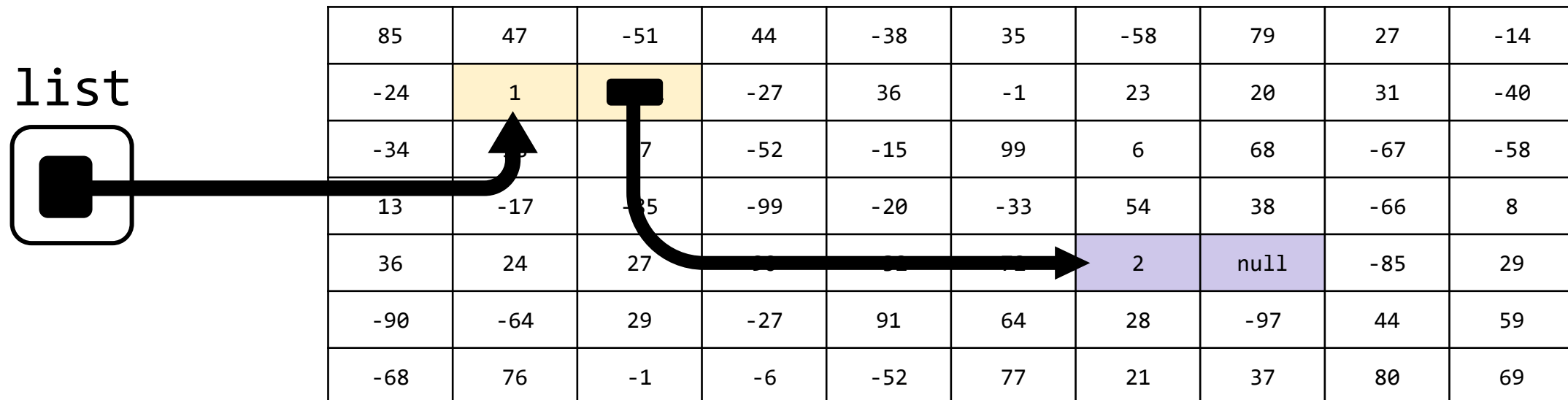
Contiguous vs. Non-contiguous: ListNode (3)

- Computer memory = one really, *really* big array.

```
ListNode list = new ListNode(1);
```

```
list.next = new ListNode(2);
```

Memory



Contiguous vs. Non-contiguous: Summary

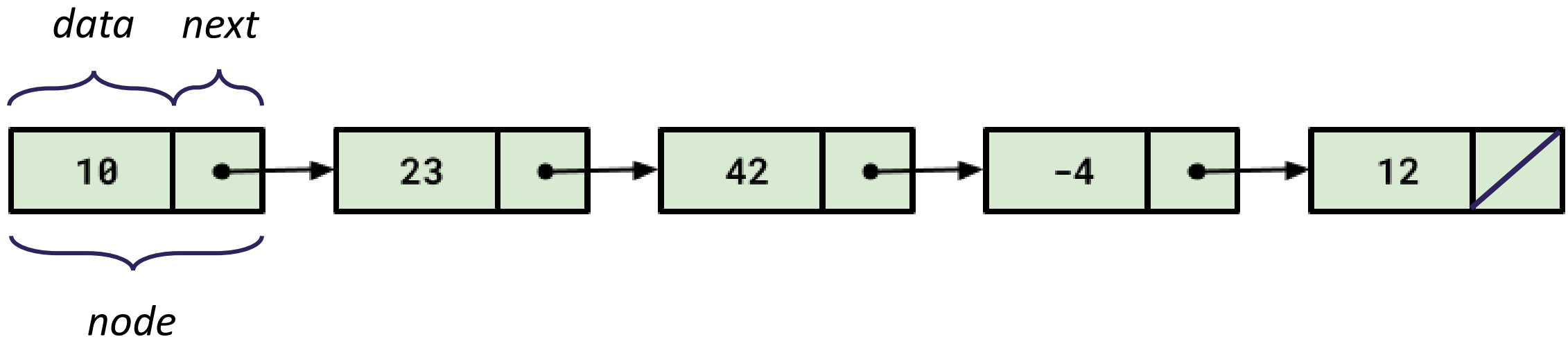
- Computer memory = one really, *really* big array.
- Contiguous memory = impossible to resize directly
 - Surrounding stuff in memory (we can't just overwrite)
 - Best we can manage is get more space and copy
- Non-contiguous memory = easy to resize
 - Just get some more memory and link it to the rest

Lecture Outline: ListNode Practice

- Announcements
- Reference Semantics Review
- Contiguous / Non-Contiguous Memory Review
- **ListNode Practice** ◀

Linked Nodes Idea

- We want to create a list of ints “**non-contiguously**”
- Accomplish this with nodes we link together
 - Each node stores an int (*data*) and a reference to the next node (*next*)



doodling