

LEC 04

CSE 123

Linked Nodes

Questions during Class?
Raise hand or send here




sli.do #cse123

BEFORE WE START***Talk to your neighbors:****How do you relax after a stressful day?***Respond on sli.do!****Instructor: Miya Natsuhara****TAs:**

Arohan	Shiven	Yuntong	Anya
Sreshta	Vrinda	Amiya	Anisha
Rushil	Gavin	Sahana	Trien
Sean B	Shreya	Anirudh	Neal
Chloe	Jonah	Rohan	Evan
Jenny	Renee	Crystal	Rena
Nate	Chris	Eeshani	
Saachi	Ishita	Prakshi	
Hawa	Kuhu	Aidan	
Maggie	Kavya	Cora	
Sean E	Misha	Nhan	

Music:  [CSE 123 26sp Lecture Tunes](#) 

Lecture Outline: Announcements

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

Announcements

- Creative Project 0 feedback will be released later today!
 - Check out the Minimum Grade Guarantee Calculator in the [syllabus](#) for tracking your grades
- Resubmission Cycle 0 opens today, closes this Friday, April 17
 - Normally resubmissions will be open Mon – Fri each week
- Programming Assignment 0 due tonight, April 15 at 11:59pm!
 - See generic [Programming Assignment rubric](#) posted on website
- Creative Project 1 will be released tomorrow, April 16
 - Focused on design and implementation of data structures
- Quiz 0 next week (Tuesday, April 21)
 - See [Quiz Logistics announcement](#) on Ed
 - Quiz 0 resources provided in [Resources tab on Ed](#) (reference sheet, practice quizzes)

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)

```
int x = 10;
int y = x;

y++; // x remains unchanged
```

```
int[] x = new int[5];
int[] y = x;

y[0]++; // x[0] changed
```

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)

```
int x = 10;
int y = x;

y++; // x remains unchanged
```

```
int[] x = new int[5];
int[] y = x;

y[0]++; // x[0] changed
```

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)

```
int x = 10;
int y = x;

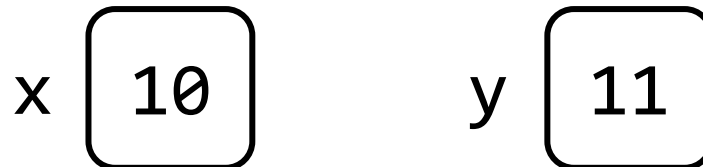
y++; // x remains unchanged
```

```
int[] x = new int[5];
int[] y = x;

y[0]++; // x[0] changed
```

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)

```
int x = 10;
int y = x;

y++;    // x remains unchanged
```

```
int[] x = new int[5];
int[] y = x;

y[0]++; // x[0] changed
```

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)

```
int x = 10;
int y = x;

y++;    // x remains unchanged
```

```
int[] x = new int[5];
int[] y = x;

y[0]++; // x[0] changed
```

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)

```
int x = 10;
int y = x;

y++;    // x remains unchanged
```

```
int[] x = new int[5];
int[] y = x;

y[0]++; // x[0] changed
```

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

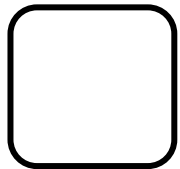
85	47	-51	44	-38	35	-58	79	27	-14
-24	-38	-66	-27	36	-1	23	20	31	-40
-34	38	37	-52	-15	99	6	68	-67	-58
13	-17	-85	-99	-20	-33	54	38	-66	8
36	24	27	90	-32	72	-73	11	-85	29
-90	-64	29	-27	91	64	28	-97	44	59
-68	76	-1	-6	-52	77	21	37	80	69

Contiguous vs. Non-contiguous: array (1)

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

Memory

arr

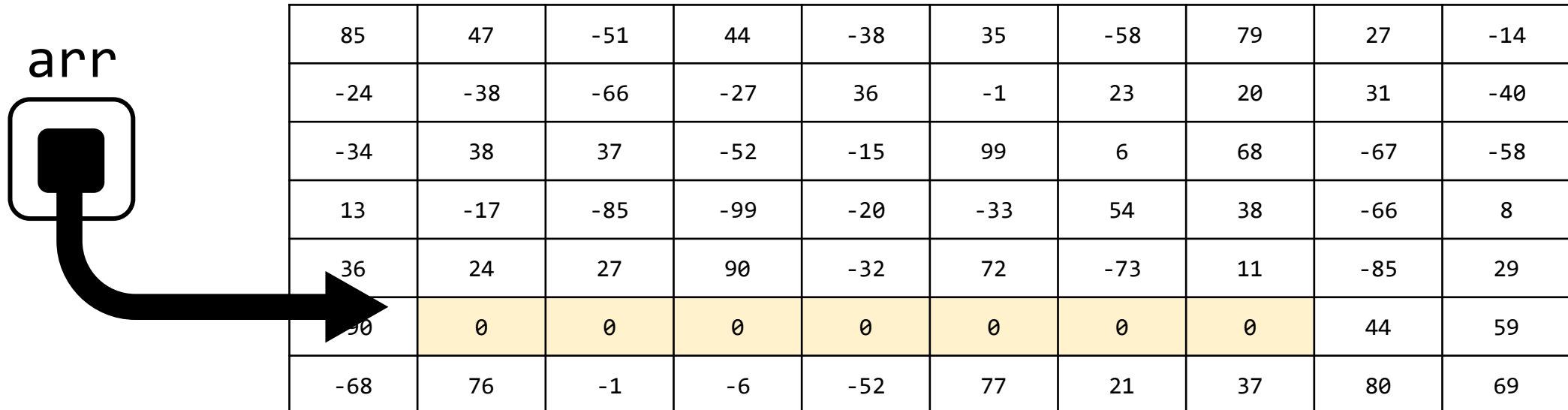


85	47	-51	44	-38	35	-58	79	27	-14
-24	-38	-66	-27	36	-1	23	20	31	-40
-34	38	37	-52	-15	99	6	68	-67	-58
13	-17	-85	-99	-20	-33	54	38	-66	8
36	24	27	90	-32	72	-73	11	-85	29
-90	-64	29	-27	91	64	28	-97	44	59
-68	76	-1	-6	-52	77	21	37	80	69

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)

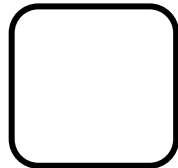
- Computer memory = one really, *really* big array.

```
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

list



85	47	-51	44	-38	35	-58	79	27	-14
-24	1	null	-27	36	-1	23	20	31	-40
-34		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

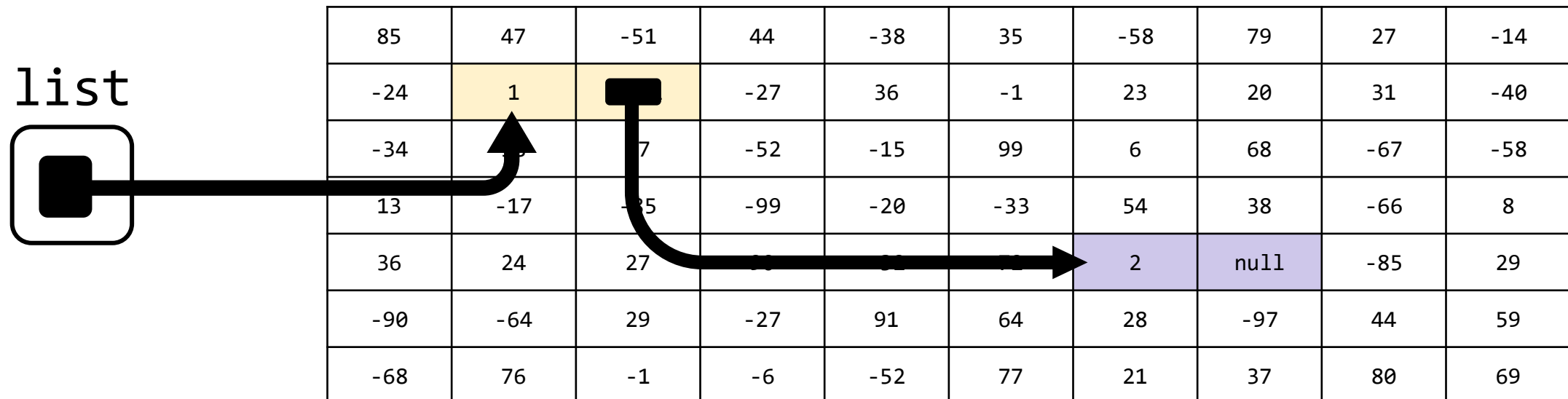
Contiguous vs. Non-contiguous: ListNode (3)

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

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

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

Memory



Contiguous vs. Non-contiguous: Summary

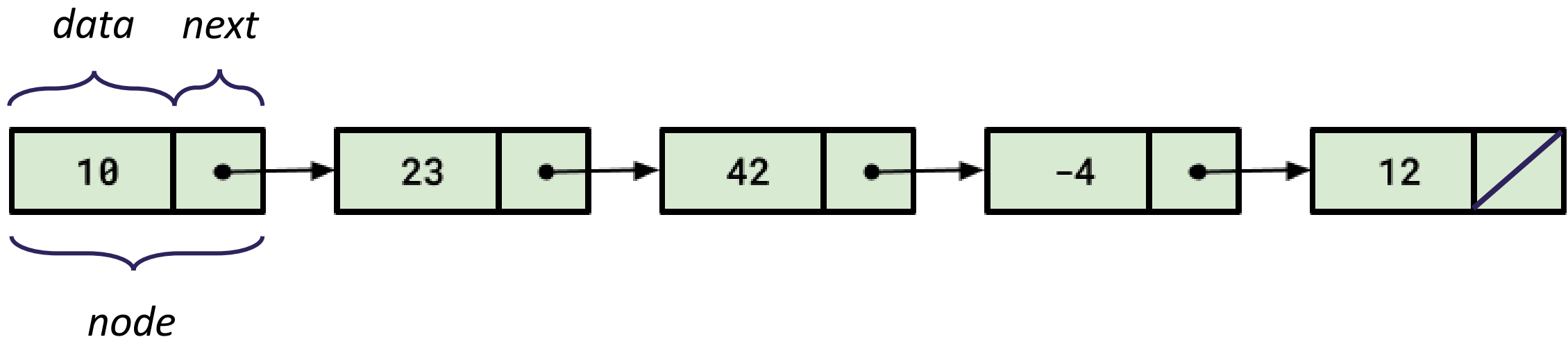
- Computer memory = one really, *really* big array.
- Contiguous memory = impossible to resize directly
 - Surrounding stuff in memory (we can't just overwrite)
 - Best we can manage is get more space and copy
- Non-contiguous memory = easy to resize
 - Just get some more memory and link it to the rest

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