CIS 110 Recitation

Linked Lists & Interfaces

July 31 2017

Agenda + Logistics

- Today:
  - Interfaces
  - Linked Lists
- Any requests?
  - Homework 4?
  - Lecture material?
- HW4 due Wednesday
- Final Exam Thursday

Linked Lists

Overview

- Linked data structure
- Made up of Nodes
- A Node stores two things:
  - 1) Data (can be anything)
  - 2) A pointer to the next Node in the list
- The last Node points to null
- Note it is a recursive structure!
  - Every Node is the start of a linked list
- Generally we keep track of the head (and maybe tail) of the list
  - We can access every other Node by following the pointers from the head

TIP: Draw it out!

Arrays

- singly typed (only one type of data)
- statically sized
- each index only has a value
- constant-time item access - O(1)

Linked Lists

- singly typed (only one type of data)
- dynamically sized
- each node stores data & next pointer
  - More memory used than an array
- linear-time item access - O(n)

Applications

When would we want to use an array or linked list in the following situations?

- grocery list
- office hour queue
- arcade game's high score
- list of students currently enrolled in 110

Looping and Linked Lists

- We only have access to the head of a list, so how do we access other things?
  - Answer: a loop!
  - Note that loop variables can be Node's, they don't have to be ints
- Tracing is very important here!
  - Let's try trace the loop below
  - Again, draw it out!
- Can manipulate the ending conditions and the update as needed

```
for (Node current = first; current != null; current = current.next) {
        System.out.println(current.data);
}
```

Exercise

Create a LinkedList that stores doubles

Steps:

1.) create a private Node class

2.) write constructor, method signatures, determine field variables, write test cases

3.) fill in functions

4.) test LinkedList

Recursion and Linked Lists

- Recall linked lists are a recursive structure!
  - A linked list of size n is made up of a Node and a linked list of size n-1
- This means we can implement functions recursively as well as iteratively
- Tips for recursion:
  - Remember to have a base case and recursive call
  - Try work it out on a simple case first
  - Test your solution!


"That's all Folks!"