Intro to Data Structures

Lecture #10 – Intro to Linked Lists
September 21, 2014
Mark Stehlik

Outline for Today

- HW3 questions?
- Q3 debrief (avg 86, max 98, min 65);
- Starting down the data structures path…
- ArrayList implementation details
- Intro to Linked Lists

Outline until Eid al Adha

- HW2 returned Tuesday
- rest of this week – Linked Lists
  - including recitation Thu
- Next week
  - Recursion (Sunday)
  - Review (Tuesday) – old midterm?

Programming Style (general comments)

- Method comments should describe algorithm

  /*

  *

  */

  – in-line (//) should note something “interesting”
- Don't always go with your first algorithm…
- Test, test, test (especially things the spec says you should)!
- Try to figure out where the error lies…

HW2 issues

- What about a findPerson() helper function?
  - signature?
  - visibility? (also, more generally…)
- updateDatabase()
  - file needed to be written so that it could be read!

Quiz 3 issues

- 0 people got Q4 perfect (11 used a loop)! Why?
- public void setPerson(Person p, int index)
  - needed if to make sure index was valid
    - which means what?
  - not this: !

```
for (int i = 0; i < numContacts; i++)  ???
        if (i == index)
            contacts [i] = p
        else
            s.o.p.("Error")
```

ArrayList methods

- From java.util.ArrayList (the ArrayList API):
  - list.add(value); //adds value to end of *list* O(1)
  - list.add(index, value); //adds value at index O(n)
  - list.remove(value); //removes first occurrence of value O(n)
  - list.remove(index); //removes element at index (slides down)
  - list.clear(); //removes all elements, *list* is empty O(1)
  - list.contains(value); //true if value in *list*; false if not O(n)
  - list.get(index); //returns the element at index O(1)
  - list.indexOf(value); //returns first index of value; -1 if not O(n)
  - list.isEmpty(); //what do you think? O(1)
  - list.set(index, value); //sets element at index to value O(1)
  - list.size(); //returns the number of elements in *list* O(1)

ArrayList implementation

- The API guarantees the big O performance of certain operations…
- What's under the hood to get that performance?
  - constant time access to an individual element implies an array!
  - what does that mean?

Linked Lists

- ArrayLists are quite useful since they remove the major issue with arrays - ArrayLists grow as you need them
- But, there are still issues (consider how much "wasted" space there is when the ArrayList grows) - there's no such thing as a free lunch
- Linked lists were the "original" answer to the bounded array issue
- Linked lists provide a programming foundation for working with other node-based data structures (i.e., trees & graphs)

Linked Lists

- Linked list is a collection of nodes
- Unlike an array/ArrayList, the nodes are not contiguous in memory
- Thus, to build a linked list of nodes, you need to know where the first node is, where the next node is…
- And, you need a way to mark the "end" (what do you think that will be?)

Linked Lists (recursive definition)

- A linked list is
  - null (the only testable reference value)
  - a node whose next field stores (points to/refers to) a linked list
  - recursion, anyone?

List Node

- Needs to store (fields/attributes)
  - data (a String, for now)
  - the location of the next node (a reference to a Node)
- Methods
  - constructor
  - two getters (one for the data, one for the next)
  - one setter (the next)

Linked Lists

- ListNode front = new ListNode("first", null);
- Let's build a small test list – to the code!
- As we work through the code, we'll be drawing lots of "box-and-pointer" diagrams. These will serve you incredibly well as a programmer; get used to drawing lots of them!