CSSE 220 Day 18

Continue Data Structures Grand Tour
Work on Hardy's Taxi

CSSE 220 Day 18

- You are to review several teams' Minesweeper programs for functionality issues before the end of the week.
  - Details in yesterday's email.
  - Surveys and instructions are also on ANGEL
- Markov assignment will be done in pairs. You can choose your partner again.
  - Must be different than your Minesweeper partner.
  - If you have not filled out the survey, please do it now.

Hardy Grading Script …

- … appears to be ready. Let me know if you have any problems with it.

```
addiator 4:53am > cd /class/csse/csse220/200820/
addiator 4:55am > ./check Hardy
Checking Hardy
Clearing
/afs/rh/class/csse/csse220/200820/turnin/mrozekma/Hardy/extract/
Copying *.java... done

Compiling project...
No compile errors found
mrozekma - Summary for Hardy
Graded on Tue Jan 15 04:55:28 EST 2008

N    Points  Your Answer
1    15/15     1729 = 1^3 + 12^3 = 9^3 + 10^3
5    18/18     32832 = 4^3 + 32^3 = 18^3 + 30^3
30   10/10     515375 = 15^3 + 80^3 = 54^3 + 71^3
100  4/4     4673088 = 25^3 + 167^3 = 64^3 + 164^3
500  3/3     106243219 = 307^3 + 426^3 = 363^3 + 388^3

Points earned: 50/50
```

Answers to your questions

- Abstract Data Types and Data Structures
- Hardy's Taxi
- Material you have read
- Anything else

Today's agenda

- Continue the Data Structures Tour
- Work on Hardy's taxi

But first …

- Look at the solution to the BinaryInteger problem from Tuesday's class.
- It will also be on ANGEL after my second class today.

Some basic data structures

- Array (1D, 2D, …)
- Stack

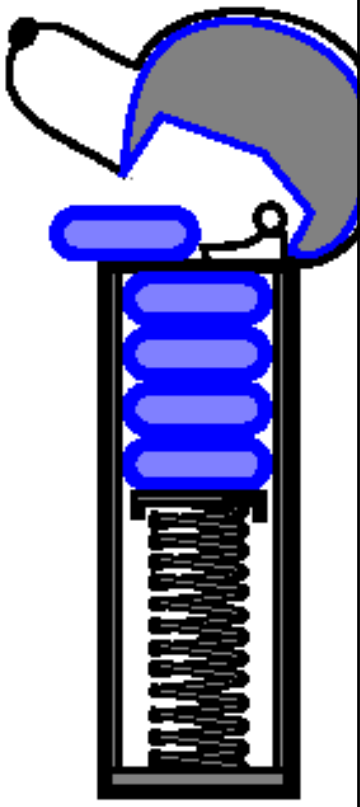
What is "special" about each data type?

What is each used for?

What can you say about time required for
- adding an element?
- removing an element?
- finding an element?

You should be able to answer all of these by the end of this course.

Stack



- Last-in-first-out (LIFO)
- Only top element is accessible
- Operations: push, pop, top, topAndPop
  - All constant-time.
- Easy to implement as a (growable) array with the last filled position in the array being the top of the stack.
- Applications:
  - Match parentheses and braces in an expression
  - Keep track of pending function calls with their arguments and local variables.
  - Depth-first search of a tree or graph.

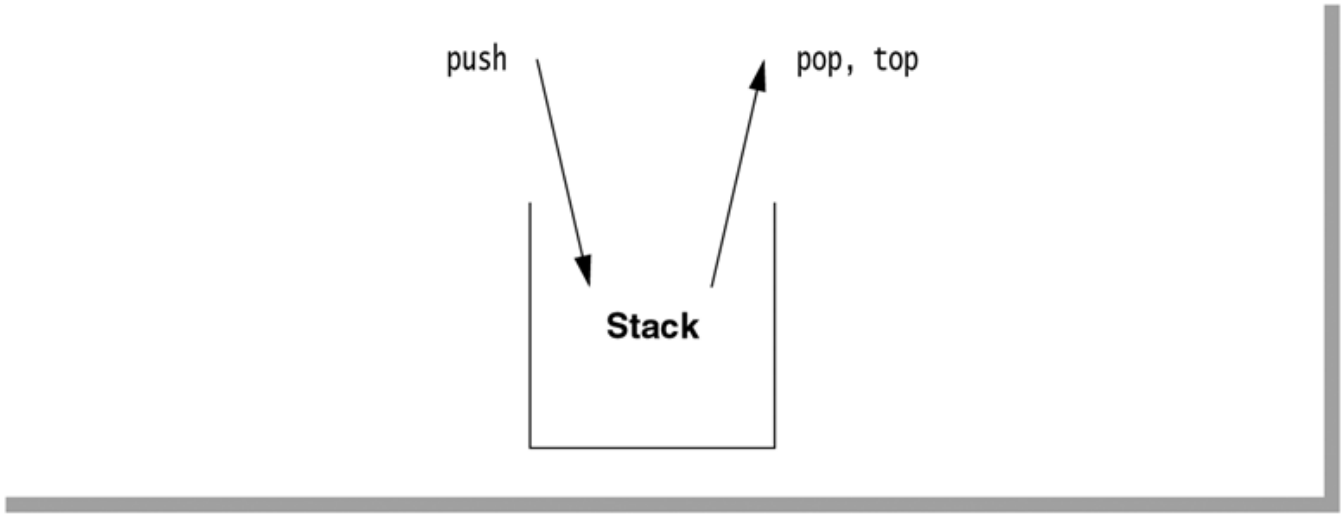


figure 6.20

The stack model: Input to a stack is by `push`, output is by `top`, and deletion is by `pop`.

Some basic data structures

- Array (1D, 2D, …)
- Stack
- Queue

What is "special" about each data type?

What is each used for?

What can you say about time required for
- adding an element?
- removing an element?
- finding an element?

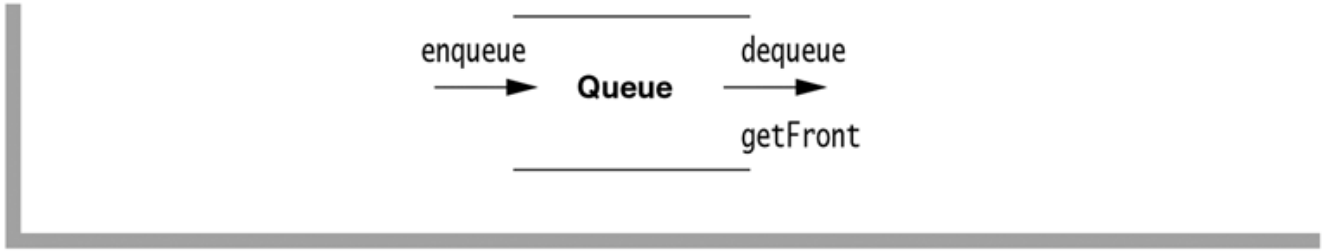
You should be able to answer all of these by the end of this course.

Queue

- First-in-first-out (FIFO)
- Only oldest element in the queue is accessible
- Operations: enqueue, dequeue
  - All constant-time.
- Implement as a (growable) "circular" array
  - http://maven.smith.edu/~streinu/Teaching/Courses/112/Applets/Queue/myApplet.html
- Applications:
  - Simulations of real-world situations
  - Managing jobs for a printer
  - Managing processes in an operating system.
  - Breadth-first search of a graph.

figure 6.22

The queue model: Input is by `enqueue`, output is by `getFront`, and deletion is by `dequeue`.



Some basic data structures

What is "special" about each data type?

What is each used for?

What can you say about time required for
- adding an element?
- removing an element?
- finding an element?

- Array (1D, 2D, …)
- Stack
- Queue
- List
  - ArrayList
  - LinkedList

You should be able to answer all of these by the end of this course.

List

- A list is an ordered collection where elements may be added anywhere, and any elements may be deleted or replaced.
- **Array List**: Like an array, but growable and shrinkable.
- **Linked List**:

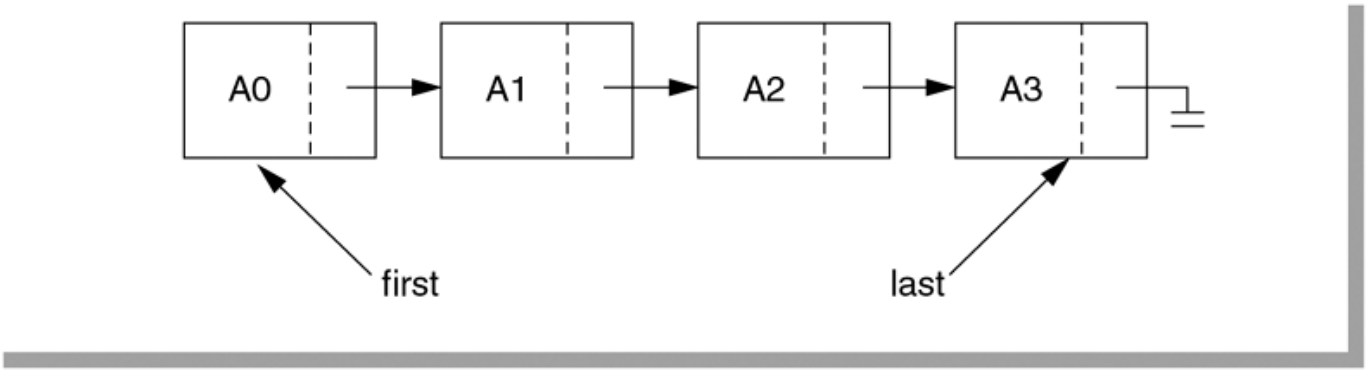


figure 6.19
A simple linked list

- Running time for add, remove, find?

List Code Example

```
LinkedList<String> list = new LinkedList<String> ();
list.add("abc");
list.add("xyz");
list.add(1, "ddd");
list.add(2, "jkl");
System.out.println(list);
list.remove("ddd");
System.out.println(list);
list.remove(2);
System.out.println(list);
```

- Output:
- **[abc, ddd, jkl, xyz]**
  [abc, jkl, xyz]
  [abc, jkl]

Some basic data structures

What is "special" about each data type?

What is each used for?

What can you say about time required for
- adding an element?
- removing an element?
- finding an element?

- Array (1D, 2D, …)
- Stack
- Queue
- List
  - ArrayList
  - LinkedList
- Set
- MultiSet

You should be able to answer all of these by the end of this course.

Set and MultiSet

- **Set**: A collection that never contains two distinct objects **a** and **b**, such that a.equals(b).
- **Multiset** (a.k.a. bag). An item can occur multiple times, and the collection keeps track of the multiplicity of each.
- Two Java representations of sets
  - TreeSet (based on a Binary Tree) – items ordered
  - HashSet (based on Hash Table) – items not ordered.
- Running times for add, remove, find?

Java Set Example

- Define a class to insert in the set:

```
class Pair implements Comparable<Pair>{
   private  String s1, s2;

   public Pair(String s1, String s2) {
      this.s1 = s1;
      this.s2 = s2;
   }

   @Override public String toString() {
      return String.format("<%s,%s>", this.s1, this.s2);
   }

   public int compareTo(Pair other){
      return this.s1.compareTo(other.s1);
   }

   @Override public boolean equals(Object other) {
      Pair oth = (Pair)other;
      return this.s1.equals(oth.s1);
   }

   @Override public int hashCode() {
      return s1.hashCode();
   }
```

Java Set Example – TreeSet

```
TreeSet<Pair> ts = new TreeSet<Pair> ();
ts.add(new Pair("abc", "1"));
ts.add(new Pair("def", "2"));
System.out.println(ts);
System.out.println(ts.contains(new Pair("abc", "3")));
ts.add(new Pair("abc", "3"));
System.out.println("After duplicate \"add\": " + ts);
ts.remove(new Pair("abc", "3"));
System.out.println(ts);
ts.add(new Pair("abc", "3"));
System.out.println(ts);
ts.add(new Pair("bbb", "4"));
System.out.println(ts);
```

Output:

```
[<abc,1>, <def,2>]
true
After duplicate "add": [<abc,1>, <def,2>]
[<def,2>]
[<abc,3>, <def,2>]
[<abc,3>, <bbb,4>, <def,2>]
```

Java Set Example – HashSet

```
HashSet<Pair> t2 = new HashSet<Pair> ();
t2.add(new Pair("abc", "1"));
t2.add(new Pair("def", "2"));
System.out.println(t2);
System.out.println(t2.contains(new Pair("abc", "3")));
t2.add(new Pair("abc", "3"));
System.out.println("After duplicate \"add\": " + t2);
t2.remove(new Pair("abc", "3"));
System.out.println(t2);
t2.add(new Pair("abc", "3"));
System.out.println(t2);
t2.add(new Pair("bbb", "4"));
System.out.println(t2);
```

Note that the elements are not in Comparable order.

Output:

```
[<abc,1>, <def,2>]
true
After duplicate "add": [<abc,1>, <def,2>]
[<def,2>]
[<abc,3>, <def,2>]
[<abc,3>, <def,2>, <bbb,4>]
```

Some basic data structures

What is "special" about each data type?

What is each used for?

What can you say about time required for
- adding an element?
- removing an element?
- finding an element?

- Array (1D, 2D, …)
- Stack
- Queue
- List
  - ArrayList
  - LinkedList
- Set
- MultiSet
- Map (a.k.a. table, dictionary)
  - HashMap
  - TreeMap

You should be able to answer all of these by the end of this course.

Map

- A Table of key-value pairs.
- Insert and look up things by key.
- Implementations:
  - TreeMap
  - HashMap
- Same running time as the corressponding sets.
- More details next time.

Work on Hardy's Taxi

- Or on HW 18 if you have finished Hardy's Taxi.