# Selecting Abstract Data Types (ADTs)

CS 61B Spring 2016: Discussion 5

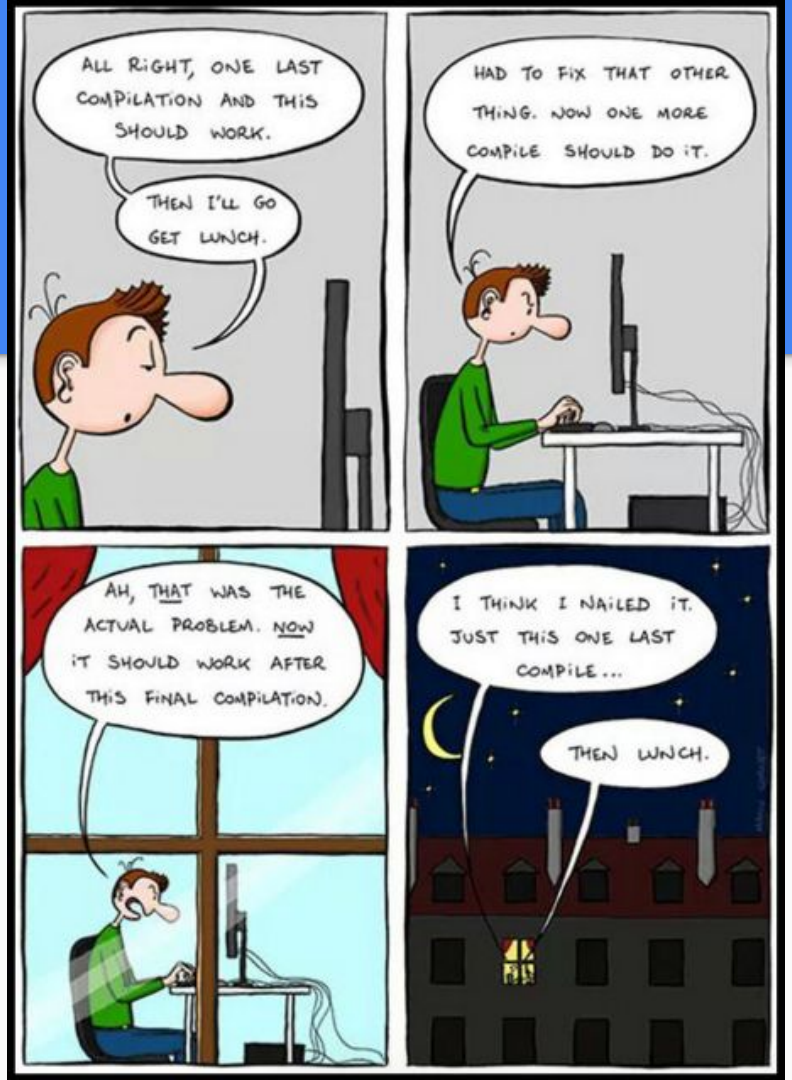
# Announcements

We are grading your exams! Almost done; definitely done before the drop deadline.

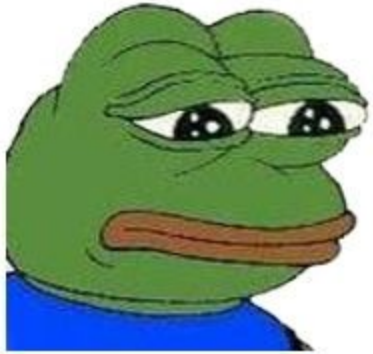
Homework 1 will be coming out soon and currently planned to be due Friday 2/26.

How was the midterm?

Thoughts on last week's discussion structure?


ALL RIGHT, ONE LAST COMPILATION AND THIS SHOULD WORK.
THEN I'LL GO GET LUNCH.
HAD TO FIX THAT OTHER THING. NOW ONE MORE COMPILE SHOULD DO IT.
AH, THAT WAS THE ACTUAL PROBLEM. NOW IT SHOULD WORK AFTER THIS FINAL COMPILATION.
I THINK I NAILED IT. JUST THIS ONE LAST COMPILE...
THEN LUNCH.

FEELS BAD MAN.



FEELS BAD.

# Abstract Data Types

Some basic abstract data types that we will work with...

- List
- Set
- Stack
- Queue
- PriorityQueue
- Map

# List

```
List {
     insert(item, position);     // inserts item into list at position
     get(position);              // returns the item in list at position
     size();                     // returns number of items in list
}
```

# Set

```
Set {
     add(item);          // puts item into the set, does not add duplicates
     contains(item);     // returns true if item is in the set, false otherwise
     items();            // returns a List of all items in some arbitrary order
}
```

# Stack

```
Stack {
     push(item);    // puts item onto the “top” of the stack
     pop();         // removes and returns the most recently put item
                         (the item at the “top” of the stack)
     isEmpty();     // returns true if the stack is empty, false otherwise
}
```

# Queue

```
Queue {
     enqueue(item);     // puts item into the “back” of the queue
     dequeue();         // removes and returns the least recently put item
                              (the item at the “front” of the queue)
     isEmpty();         // returns true if the queue is empty, false otherwise
}
```

# PriorityQueue

```
PriorityQueue {
     enqueue(item, priority);     // puts item into the queue with a priority
     dequeue();                   // removes and returns the item with the highest
                                       priority
     peek();                      // returns the item with the highest priority
}
```

# Map

```
Map {                    // similar to dictionary in Python
     put(key, value);    // puts key into the map and associates it with value;
                              if key already in map, replace existing value with
                              given value
     get(key);           // returns value associated with key
     keys();             // returns a List of all keys in some arbitrary order
}
```

# Let’s try Problem 2!

# Problem 2.1

Q. Given a news article, find the frequency for each word used in the article.

A. Use a map. When you encounter a word for the first time, put the key into the map with a value of 1. For every subsequent time you encounter a word, get the value, and put the key back into the map with its value you just got, plus 1.

# Problem 2.2

Q. Given an unsorted array of integers, return the array sorted from least to greatest.

A. Use a priority queue. For each integer in the unsorted array, enqueue the integer with a priority equal to its value. Calling dequeue will return the largest integer; therefore, we repeatedly call dequeue and insert the values in backwards order (from index length-1 to 0) into our array. Then, return this newly filled in array.

# Problem 2.3

Q. Implement the forward and back buttons for a web browser.

A. Use two stacks, one for each button. Each time you visit a new web page, add the previous page to the back button's stack. When you click on the back button, add the current page to the forward button's stack, and pop a page from the back button's stack. When you click on the forward button, add the current page to the back button's stack, and pop a page from the forward button's stack. When you visit a new page (i.e. not something from either stack), clear the forward button's stack.

# Now for Problem 3!

# Problem 3.1

```
BiDividerMap {
    put(K, V);          // put (K, V) pair
    getByKey(K);        // get value for K
    getByValue(V);      // get key for V
    numLessThan(K);     // return number of keys less than K
}
```

# Problem 3.1

Create two maps: one maps keys to values (K->V), the other maps values to keys (V->K). Then, whenever you call **put(K, V)** in BiDividerMap, you also have to **put(K, V)** and **put(V, K)** into your two component maps.

In **numLessThan(K)**, get a list of keys, sort it, then iterate and count until you find a key greater than **K.**

# Problem 3.1

**Improvement:** Add another map that maps keys to their “rank” (how many keys are less than them) plus a boolean that represents whether this rank is cached (up-to-date).

**key -> [rank, cached]**

When **put** is called, set cached=false for that key. When **numLessThan** is called, if cached is true, return the value of rank for that key. Otherwise, rebuild this map and then set cached=true.

# Problem 3.2

```
MedianFinder {
    add(int x);         // adds integer x into collection
    getMedian();        // return median integer from collection
}
```

# Problem 3.2

Use a list.

When you **add**, insert into the back of the list (nothing special here).

When you **getMedian**, sort the list, compute the size of the list (or even better, maintain the size of the list as we did in Project 1), then return the middle item (by indexing into our list at the right index).

# Onto Problem 4...

# Problem 4

In a nutshell: build a queue using only stack(s). You have access to this Stack class:

```
public class Stack {
     public Stack() {
          // Instance variables get initialized here...
     }
     public void push(int item) { ... }
     public int pop() { ... }
     public boolean isEmpty() { ... }
}
```

# Problem 4

```
public class SQueue {
	private Stack inStack;
	public SQueue() {
		inStack = new Stack();
	}
	public void enqueue(int item) {
		Stack tempStack = new Stack();
		while (!inStack.isEmpty()) {
			tempStack.push(inStack.pop());
		}
		inStack.push(item);
		while (!tempStack.isEmpty()) {
			inStack.push(tempStack.pop());
		}
	}
```

```
public int dequeue() {
	return inStack.pop();
}
```

See you next week! :)