CS2110 Preparing for Prelim 2. Thursday, 11/21 2013

(1) 5:30–7:00 Kimball B11 **(2) 7:30–9:00 Kimball B11**
(3) 5:30–7:00 Upson B17 **(4) 7:30–9:00 Upson B17**

Review session: Sunday, 17 November, 1:00 - 3:00: Hollister B14

Student-id odd and student-id/2 odd: Exam (1) above
Student-id odd and student-id/2 even: Exam (3) above
Student-id even and student-id/2 odd: Exam (2) above
Student-id even and student-id/2 even: Exam (4) above
Conflict or disability? Contact Amy ahf42@cornell.edu

This handout explains what may be tested on prelim 2. The course website contains several prelims from past semesters. To prepare for the prelim, you can (1) practice writing Java programs/method in Eclipse, (2) read the text, (3) memorize definitions, principles, and (4) study past prelims, which can be found on the course webpage.

The overall length and balance of the exam will be similar to past prelim 2s, but the exam will cover only topics covered through Thursday, 14 November. Ignore past prelim 2 questions that touch on topics not listed below.

You are expected to know everything that was required for Prelim1. Look at the review handout for Prelim1, which can be found on the course website.

Here are some topics that might be covered:

1. Proofs. We will ask you to prove things (that an algorithm will behave correctly, or that some sort of simple closed form equation is the correct solution to an iterative equation). Familiarize yourself with the proof techniques covered in class, including the way we proved that the various graph algorithms are correct and the concept of a (weak) inductive proof. We will expect more rigorous arguments for correctness, big-O complexity, correctness of recursive procedures, etc.

2. Algorithmic complexity. Big-O complexity notation and the associated definitions. You should understand how to derive a big-O complexity formula for an algorithm, best-case/worst-case/average complexity, the notion that what this counts is some sort of "operation we care about" and not every line of code, etc.

3. Abstract data types (ADTs) and how they can be defined in Java (using an Interface).

3. Searching and sorting. We have covered a number of sorting algorithms and you need to know them! Linear versus binary search, mergesort, quicksort, heapsort. How they work, complexity, data structures they use. You should be able to write mergesort and quicksort, using high-level statements for the parts that actually massage the array (e.g. "merge sorted partitions b[h..k] and b[k+1..n]"). Make sure you understand the min-heap data structure and the way it can be used to implement a priority queue.

4. Hashing. Understand hashing as presented in recitation.

5. Interfaces. Review the interface lecture materials and make sure you understand the ideas. Be familiar with the standard operations that are supported by common data structures implementing Collection<T>, List<T>, Set<T>, Map<T>, ArrayList<T>, etc.

6. Graphs. Know about depth-first and breadth-first search, Dijkstra's shortest path algorithm, spanning trees, Kruskal's and Prim's algorithm. You can expect questions that involve graphs: be ready to tell us which algorithm is the best choice for solving a problem, precisely what that algorithm does, why it would solve a problem, and how costly it might be. You will *not* be asked to provide code for these standard algorithms: e.g. you can say "use DFS" without coding DFS on the exam.

7. GUIs. You will be not be asked to write GUI programs, although we might ask you read and understand small ones. At this point you should know about the three major classes that can contain components (JFrame, JPanel, and Box), what their layout managers are and how they lay out components on the screen, and how one listens to events.

8. Keep in mind the following.

A. Being able to write correct code in Java critical. We will continue to have a large number of points on coding questions. We plan to grade them with a bit more insistence on correct Java. We were relaxed about giving partial credit for code in Matlab or Python on prelim1. Don't expect a second "free pass" on that on prelim2. Our graders will even catch errors like confusion between instance and static methods, syntax errors, unnecessarily complex code, etc. On prelim1, you got full credit if your code was correct. On prelim2, you might lose credit for code that is long, is inefficient, or reveals a poor grasp of Java features.

B. We expect you to know Java —not just the bits and pieces of Java used on slides in class. If there is some aspect of Java that worries you, read about it in Appendix A.

C. Use the powerful built-in Java tools. We give maximum credit for concise, elegant code that doesn't reinvent the wheel. Know how to use standard Java types like ArrayList, HashSet, HashMap, and know the preexisting methods available for Collections, Arrays, Strings, etc.