Discrete Structures
CMSC 250

Course Overview, Essential Information re: exams, grades, etc.

Fall 2013

1. Introduction
   - Administrative Matters
   - Motivation
   - Some terms and topics, defined
   - Typical application areas
   - Overview of topics presented

2. Instructional objectives, course desiderata
   - Expectations

Important Resources

- (Nearly) Every week expect
  - a Quiz
  - a Homework assignment; and possibly
  - a Worksheet.
- Check Canvas and Piazza regularly;
- 2 (two) midterms are given for this course; check the Syllabus for details.
- Every student is expected to have read and will attest to the University's Honor Code on submissions.

Why you will care

Why should I care about any of this? I already know how to program!

- The kinds of mathematical concepts and objects that we will explore in this class *simplify* and *organize* a great number of topics that might otherwise appear unrelated.
- Computer Science grew out of modern (early twentieth century) mathematics and engineering.
- Mathematics *might not be* what you think!
- What you learn here will enable you participate in the broader discussions of mathematics and computer science.

What is the course about anyway?

Discrete Structures

- "Discrete" objects . . .
  - Are *enumerable*, meaning that they can be counted like integers;
  - May (often) be composed to create more interesting structures;
  - Effectively and naturally capture a spectrum of behaviors essential to all of computing;
  - Allow for a certain type of reasoning that is constructive and powerful.

"Continuous" is the (near) opposite of "discrete." Most of your exposure to mathematics has been with real numbers, without regard to structure, properties, or proof. This is about to change.

. . . and "structures," suggest . . .

- Things that have or are recognizable by their *properties*: Now, what could this possibly mean in a "math" class"?

. . . and "structures," suggest . . .

- Things that have or are recognizable by their *properties*: Now, what could this possibly mean in a "math" class"?
- Shift focus from an *operational* treatment ("plug and chug") to a structural understanding: conjecturing, proving, and making new meanings.

. . . and "structures," suggest . . .

- Things that have or are recognizable by their *properties*: Now, what could this possibly mean in a "math" class"?
- Shift focus from an *operational* treatment ("plug and chug") to a structural understanding: conjecturing, proving, and making new meanings.
- Our plan: learn how to reason mathematically: relying on logic, axioms, and creative thinking.

Some specific application areas . . .

We often will relate this material to "real-world" applications:

- Logic, in general, pervades all of computer science, from understanding the simplest "if-then" statements to the design of large, complex systems.
- Propositional logic is directly applicable to the design of circuits;
- Mathematical principles, such as factorizations, inverses, and induction, appear in many high-profile applications—such as computer architectures, cryptography, and the design of algorithms.
- Theorem-proving is learning how to program.

A tentative overview of Course Topics

- Propositional logic (as applied to circuits), and predicate calculus—quantification and patterns of mathematical proof.
- The reading and writing of mathematical proofs in a variety of settings–with an emphasis on inductive proofs as these are relevant to computer science.
- Topics from elementary number theory, binary arithmetic and circuits, inductive structures and recursion.
- Topics from algebra: sets, counting, including combinations and probability, functions, including cancellation properties and inverses, as well as binary relations.
- (Time permitting) An introduction to Graph Theory.

Speaking directly to you . . .

The more code that I wrote, the more I realized that a robust grasp of, and a continuing curiosity about these mathematical principles made me just better at what I did, and, more importantly, *happier* in my career.

Coding and contracts

- Writing complex software is akin to constructing contracts.
- Our contractual obligation to you:
  - Organize and present these materials.
  - Try to understand your questions and concerns;
  - Make the abstract real . . . and vice versa
- Your obligations (in addition to the University's requirements):
  - Attend class;
  - Ask "what" and "why" about mathematical objects as often as you ask "when" and "how."
  - Believe in yourself!