CSCI/MATH

243

Discrete Mathematics and Functional Programming

Spring 2017 MFW 2:15-3:25 pm SCI 131

http://cs.wheaton.edu/~tvandrun/cs243

Thomas VanDrunen

☎630-752-5692 630-639-2255 ✉Thomas.VanDrunen@wheaton.edu

Office: SCI 163 Office hours: MTuWF 3:30–4:30pm; Th 9:30–10:30am and 1:00–3:00pm.

Contents

Catalog description. Sets, logic, the nature of proof, induction, algorithms, algorithm correctness, relations, lattices, functions, and graphs. Functional programming and recursion using the ML programming language.

Textbook. Thomas VanDrunen, *Discrete Mathematics and Functional Programming*, Franklin, Beedle and Associates, 2013.

Objectives. The chief goal of this course is to teach you formal reasoning, practiced under two heads: mathematical proofs and computer programs. At the end of this course you should be able to

- Manipulate symbolic logical forms.
- Write mathematical proofs, especially for results from basic set theory.
- Write simple programs in the ML programming language.

This course also fulfills the AAQR thematic core tag in the Christ at the Core curriculum and supports the program outcomes of the computer science major. In terms of the AAQR learning outcomes and the computer science program outcomes, at the end of this course you should be able to

- Capture and model phenomena in nature, culture, and the human-built world using sets, relations, and functions as well as ML datatypes. (AAQR LO # 1, CSCI PO B & C)
- Devise, implement (in the ML programming language), and test solutions to algorithmic problems, using symbolic logic (especially quantification) to analyze the problem and synthesize a solution. (AAQR LO #2, CSCI PO A & C)
- To write formal proofs for propositions about sets, relations, functions, and the correctness of algorithms. (AAQR LO #3, CSCI PO C & E)

Other themes include

Writing and using formal definitions. We look carefully at how to define formal, rigorous definitions of mathematical ideas, built from primitive terms.

Thinking recursively. *Recursion* is defining something in terms of itself. This technique is crucial both to programming and to some kinds of mathematical definitions and proofs.

Analysis and synthesis. Many of our proofs and programs comprise two main steps: breaking something apart and putting something else together.

Outline. This course is organized under the following headings:

Set and List. We meet the basic mathematical concepts of set, element, set operations, cardinality, Cartesian products, and powersets. We begin the basics of the ML programming languages including functions and datatypes. We learn to use ML's main composite type, the list.

Proposition. We explore the system of boolean logic (the "first-order predicate calculus"). This heading is characterized by three "games" to exercise your understanding of symbolic logic: 1. verifying logical equivalences; 2. verifying argument forms; 3. verifying argument forms with quantification. We also write ML programs that use boolean operators and consider how the quantification of a program specification affects the algorithm to solve it.

Proof. This is the turning point of the semester, perhaps the most important heading. We learn to write careful mathematical proofs of set-theoretical propositions. This includes one of the most challenging sections, proofs about powersets. We also consider the connections between proofs and algorithms.

Relation. We build on our understanding of sets to consider the definition of mathematical relations and their properties, propositions about them, and programs that manipulate them. Relations are useful concepts in themselves, but this heading also gives us opportunity to practice further the proving and programming techniques from earlier in the course.

Self reference. Earlier parts of our study will have introduced recursive definitions, but here we take the idea head-on. Specific topics are recursive types in ML programming and proofs using structural and mathematical induction.

Function. We study functions as mathematical objects built on set theory, as we will have done for relations. The proofs in this section are an apex of the mathematical topic stream. We also learn idioms in ML programming based on the theory of functions.

Other. At the end of the semester, we will briefly cover a few extra (but important) programming topics.

For a detailed outline, see the table of contents in your textbook. For a schedule, see the course website.

Course procedures

HOW WE DO THIS COURSE. This course has a pretty predictable rhythm to it. Class time is mainly for working out new kinds of problems together. There will be daily assignments. Tests come when we get to good stopping points.

Before class you are to read the assigned sections from the textbook and, sometimes, watch a video presentation. In class I will review and highlight material, especially definitions, that you have read. I will avoid lecturing, preferring to devote most of our meeting time to practicing sample problems. Some class periods will also include demonstrations of new ML features. At the end of each class I will assign problems from the book, which will also be posted on the course website. These will be a mix of pencil-and-paper exercises for the math portion (mostly proofs) and coding exercises for the programming portion (the latter to be turned in through a web interface).

READINGS. It is important that you read the assigned sections for each day. The readings fit into three categories.

Read carefully means that I will not cover that material in class at all. It's background stuff for what we really want to talk about. You solely are responsible for it.

Read means that I will highlight and review the main ideas but not lecture on them. I will assume you have seen them before (that is, that you have read the material). We will work on sample problems from those sections in class.

Skim means that this is very difficult material that most students will need to see twice to understand. Familiarize yourself with it first, and then I will lecture on it in class. Note well that *ski**m*** does not mean *ski**p***.

VIDEOS. This semester I am going to experiment with video tutorials. Occasionally I will make a video of me explaining some of the concepts of the current topic. This is to supplement the textbook and to allow for more time for active learning in class (by cutting down on lecture and demo time). Watching the videos, along with doing the readings, will be part of class preparation.

PROJECT. This course has a term project in which you apply the discrete math material and what you have learned about programming to model real-world information or phenomena using ML. Work on the project will be spread throughout the semester. Details of the assignment can be found on the course website.

GRADING. There will be three tests (scheduled for Feb 10, Mar 20, and Apr 19, subject to change) and a final (Thursday, May 4 at 8:00 am).

instrument	*weight*
Homework	20
Project	10
Test 1	15
Test 2	15
Test 3	15
Final exam	25

I will also give one point of extra credit (applied towards homework) for every mistake you find in the textbook, if you are the first to discover it. Bigger suggestions about the presentation (like new exercises and examples or ways to make a section more understandable) will be rewarded appropriately.

HOW TO SUCCEED IN THIS COURSE. By this point in your academic career you should have developed good study habits and found what works best for you. In my experience, however, its seems many students could still use a few pointers.

Prepare for class. Set aside time the day before or in the morning to think about what we will be covering. Take the readings seriously. Try some of the exercises in the sections before we cover them in class.

Take the right amount of notes. You need to be active in class, working through the problems we're doing on the board. That said, some of you need to go easy on the note-taking. I feel sorry for the students who seem to think that their main task in class is to transcribe everything written on the board; they make themselves so busy writing, they don't have time to process what's going on in class. I wrote the textbook in a way that should minimize (but not eliminate) the need to take notes. I'd rather you put your energy into *thinking.*

Keep up with the material. The material in this class keeps on building on itself. If you don't understand something, don't just shrug it off and move on—get it right. You can use the re-turn-in policy for assignments (see below) to get credit for this. Even if it doesn't seem like last week's material is being used this week, last week's material is going to come back later.

When all else fails, *ask for help.* A lot of learning in a class like this happens during office hours.

Policies etc

ACADEMIC INTEGRITY. Students are encouraged to discuss homework problems and ideas for solutions. However, your solutions, proofs, and programs must be your own. If you are having trouble debugging a program you have written, you may show it to a classmate to receive help; likewise you may inspect a classmate's incorrect program to give help. However, you should not show *correct* code to a classmate, nor should you look at another student's correct code, to give or receive help. Homework on which students have unfairly collaborated will not be accepted.

ASSIGNMENTS. Unless otherwise specified, assignments are due at the class period after it was assigned. I will collect the assignments at the end of class. However, you are granted an automatic grace period until 5:00 pm that day. Assignments not complete by class time can be put in the instructor's box. If you have not completed the assignment by the end of the grace period (5:00 pm), then turn in what you have at that time for partial credit. Assignments are spot-checked: depending on the assignment, around half of the problems will be graded, and you won't know ahead of time which ones. (This is for practicality—the TAs and I don't have time to grade every problem.)

Unless otherwise specified, coding problems are to be turned in using the automated grader at `http://cs.wheaton.edu/~tvandrun/cs243/ml-grading.php`. The same rules about due dates and turn-in times apply. The system will run the submitted code against a series of test cases and will report on the results. (You will not see the test cases themselves. Testing your code with your own test cases is part of the assignment.) You may submit as many times as you want until the deadline. Submissions that do not pass all the test cases will be assigned partial credit by the instructor or TA, reported by email.

RE-TURN-IN OF ASSIGNMENTS. After you receive your graded assignments back, you may redo any proof, programming problem, or "game" problem (from Chapter 3) that you did not receive full points on and turn it back for regrading no more than two class meetings later. The regraded problems will be evaluated by the same criteria as originally used, and the student may earn back up to full credit for those problems. The same policy applies to regraded problems when they are turned back: if the student does not receive full credit on a re-turned-in problem, he or she may try again (indefinitely). For any re-turn-in, please also include the graded original (and any subsequent graded attempts).

Some details: For paper assignments, "two class meetings" means at the end of class two class days later. For example, if an assignment is turned back on Monday, the student must re-turn-in the assignment by the end of class Friday. Note that the end of class, not 5:00pm, is the deadline for re-turn-ins; the daily grace period does not apply. Students do not receive extra time to redo problems if they are delayed in receiving the graded assignment because of absence or lateness—time is measured by when you would have received it back, not when you actually did. Since programming assignments are submitted electronically and graded automatically, students have opportunity to re-turn-in a problem only if they have submitted a good-faith attempt by the original due date. The instructor or TA will send them a partial-credit assessment by email. Students then have three weekdays (72 hours) while the college is in session from the time that the TA emails an assessment of partial credit to re-turn-in a solution to the problem. For example, if a student receives a graded homework by email at 2:16 am Thursday, the student must re-turn-in the assignment no later than 2:16 am Tuesday. These are all accounted on a per-problem basis. Any problem for which the "two class days" period has elapsed is no longer eligible for re-turn-in; similarly, problems for which no attempt was turned in for the original due date are not eligible for re-turn-in (students may still turn in such problems for correction and comments, but not for credit). In any case, the opportunity applies only to answers that have an error of substance; answers with only a minor mistake that is completely corrected by the grader's comments may not be resubmitted (for proofs, this would apply to answers with only .25 point deduction).

ATTENDANCE. Students are expected to attend all class periods. It is courtesy to inform the instructor when a class must be missed.

EXAMINATIONS. Students are expected to take all tests, quizzes, and exams as scheduled. In the case where a test must be missed because of legitimate travel or other activities, a student should notify the instructor no later than one week ahead of time and request an alternate time to take the test. In the case of illness or other emergency preventing a student from taking a test as scheduled, the student should notify the instructor as soon as possible, and the instructor will make a reasonable accomodation for the student. Theinstructor is under no obligation to give any credit to students for tests to which they fail to show up without prior arrangement or notification in non-emergency situations. The final exam is Thursday, May 4, at 8:00 AM. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements.

SPECIAL NEEDS. *Institutional statement:* Wheaton College is committed to providing reasonable accommodations for students with disabilities. Any student with a documented disability needing academic adjustments is requested to contact the Academic and Disability Services Office as early in the semester as possible. Please call 630.752.5941 or send an e-mail to `jennifer.nicodem@wheaton.edu` for further information.

My own statement: Whenever possible, classroom activities and testing procedures will be adjusted to respond to requests for accommodation by students with disabilities who have documented their situation with the registrar and who have arranged to have the documentation forwarded to the course instructor. Computer Science students who need special adjustments made to computer hardware or software in order to facilitate their participation must also document their needs with the registrar in advance before any accommodation will be attempted.

Gender-inclusive language. The college requires the following statement to be included on all syllabi: *For academic discourse, spoken and written, the faculty expects students to use gender inclusive language for human beings.*

Office hours. I try to keep a balance: Stop by anytime, but prefer my scheduled office hours. Any time my door is closed, it means I'm doing something uninterruptible, such as making an important phone call. Do not bother knocking; instead, come back in a few minutes or send me an email.

Dress and deportment. Please dress in a way that shows you take class seriously—more like a job than a slumber party. (If you need to wear athletic clothes because of activities before or after class, that's ok, but try to make yourself as professional-looking as possible.) If you must eat during class (for schedule or health reasons), please let the instructor know ahead of time; we will talk about how to minimize the distraction.

Electronic devices. Please keep laptops and all electronic devices put away and silenced during class. (That's right, this is a computer science course, but you're not allowed to use a computer during class. Trying out programming concepts on your own during class time (for example) is not productive because it takes you away from class discussion. You cannot multi-task as well as you think you can.) ***Text in class and DIE.*** I take this very seriously.