

CIS 252 — Introduction to Computer Science — Spring 2008

Course Syllabus

JANUARY 14, 2008

Catalog Description

Programming emphasizing recursion, data structures, and data abstraction. Elementary analysis of and reasoning about programs. Public policy issues. Extensive programming. Three hours of lectures and one hour of computer laboratory.

Instructor Information

Jim Royer (royer at ecs dot syr dot edu)

Office: CST 3-132, x1028

Office hours: Tuesday 2:45–3:30pm, Friday 10:00pm–11:15pm, & by appointment.

TA Information

Paul Talaga (pgtalaga at syr dot edu)

Office: CST 4-206F

Office hours: TBA

?? ?? (?? at syr dot edu)

Office: CST ?-???

Office hours: TBA

Lab Sessions

At least one of us will always be available during scheduled lab sections:

Tuesday 12:30–1:25pm in CST 1-214

Tuesday 3:30–4:25pm in CST 1-214

Wednesday 10:35–11:30am in CST 1-214

Course Web Page

<http://www.cis.syr.edu/courses/cis252>

Textbooks and Other References

- Simon Thompson, *Haskell: The Craft of Functional Programming*, 2/e, Addison-Wesley, 1999.
- Jerry Peek, Grace Todino-Gonguet, and John Strang, *Learning the Unix Operating System, Fifth Edition A Concise Guide for the New User*, 5/e, O'Reilly, 2001.
- Other notes will be distributed during the semester as needed.

Prerequisites

MAT 295 (*Calculus I*) is a co-requisite for this course, primarily as an indicator of mathematical maturity. If you have not already passed MAT 295 and are not taking it this semester, you should come speak to the instructor to determine whether or not you are ready for this course.

In addition, PHI 251 (*Logic*) is an unofficial co-requisite for this course. We will use the connections between logic and Haskell programs throughout the semester.

Course Topics

Unix basics: basic shell commands and tools, environments, man pages, pipes, filters. The program-design recipe. Haskell basics. Higher-order functions and currying. Types, including tuples, lists, functions, and algebraic data types. Simple type checking and type inference. Polymorphism and overloading. Data and type abstraction. Historical context for various Computer Science concepts.

Course Objectives

As its name suggests, this course introduces many of the core concepts of computer science, as well as touching on a few pieces of its history.

Broadly speaking, the practice of computer science is about data structures and supported operations on them, small (or large) languages and environments, and component-based problem solving. A computer scientist must continually choose among alternatives: to do so successfully requires exposure to different options, knowledge of how to make wise choices, and experience. An important goal of this course is to expand your view of the computing landscape by helping you explore and experiment with options beyond what you've already seen.

We will be using the programming language Haskell for this course. Haskell is especially well suited for the two main themes we want to emphasize: *patterns* and *problem solving*.

Haskell supports working with patterns in two ways. First, Haskell's type system and extensive pattern-matching facility help a programmer write code that reflects the structure of the data it operates on. Second, using higher-order functions helps programmers capture and re-use a variety of computational patterns in a very succinct way.

Haskell also supports a view of component-based design that is especially well-suited for problem solving and for logical analysis. Consequently, you should begin to think about programming in a different, more systematic and logical way, even as you move on to imperative and object-oriented programming.

You'll be tempted to call this "the Haskell course". Our goal is that, by the end of the semester, you will be able to see below the surface of Haskell and recognize core concepts that you will take with you to other courses and into your professional life.

Outcomes

After completion of the course, you should be sufficiently proficient with Unix, Emacs, Haskell, and the program-design process that you can write, edit, execute, test, and debug moderate-sized programs. In addition, you should have an understanding of the Unix shell as a programming language, of Emacs as a useful program-development environment, and of Haskell as a convenient prototyping tool.

More specifically, you should be able to do the following:

Unix and Emacs

- Use the Unix command line and other features to perform basic tasks
 - Managing files and directories
 - Changing file permissions
 - Checking and removing files from a print queue

- Printing files (both postscript and non-postscript)
- Checking available options for a Unix program
- Altering one’s environment and shell via the `~/.login` and `~/.cshrc` files
- Writing one-line programs using basic pipes and filters
- Use Emacs as a program-development environment
 - Configuring its behavior via the `~/.emacs` file
 - Using apropos to locate available Emacs functions

Program Design & Development

- Recall the 4 steps of Polya’s problem-solving method.
- Recall the 5 basic components of the program-design recipe.
- When given an informal English specification of a problem, extract a relevant collection of abstract types from that specification (i.e., the *contract*).
- When given a specification of a program, construct concrete examples that satisfy that specification.
- When given a problem specification, recognize, comprehend and apply the concepts of pattern matching and higher-order functions when designing your programs.

Haskell fundamentals

- When given a Haskell expression, determine whether or not it is well-typed.
- When given a Haskell expression that is well-typed, give its type.
- When given a Haskell expression that is ill-typed, explain why it is ill-typed.
- When given a Haskell type expression, give some Haskell expressions or values of that type.
- Use higher-order functions to map, filter, and fold operations across recursively defined data (e.g., lists)
- Use Haskell patterns to write data-directed programs.
- Define Haskell algebraic types to represent abstract types, and write Haskell programs that operate over those types.
- When given a moderate-sized Haskell program and relevant input, calculate the result of that program.
- When given an informal but fairly precise English description, write a Haskell program that accurately captures the desired behavior.

Computer Science Concepts

- When given a Turing machine description and a specific input string, be able to calculate the resulting output string.
- When given an English description of a problem, write a Turing machine that accurately captures the desired behavior.

Outcome Measurements

Your final grade will be based on a variety of activities:

- Homework assignments (30% of final grade)

There will be a homework assignment about every week: The lowest homework grade will be dropped at the end of the semester. Occasionally, students may be asked to explain their homework to me or to the TA: in such cases, the homework grade will be based on the results of this explanation.

- Labs (15% of final grade)
There will be a lab assignment about every week: The lowest lab grade will be dropped at the end of the semester.
- Quizzes (55% of final grade)
There will be six in-class exams during the semester. There will also be a two-hour *optional* final exam: the test portion of your final grade will be the greater of (1) your cumulative average of the in-class quizzes, and (2) your score on the final exam.

You should also be aware of the following **submission practice and policy**:

On some homework and lab assignments, you may work singly or in pairs; on other assignments, you must work alone. If you work with someone else, you should submit a single solution: each person's name must appear clearly on the first page. All members of a group will receive the same grade (even if only one student is asked to explain a homework); groups of more than two will receive **no credit**.

Labs and homework assignments should be placed in the marked bin in CST 3-212 (the Foundry); you will also need to submit code electronically (**details are available** on the class web site). Assignments are due by the date and time specified on them: *No late assignments will be accepted.*

Ethics

I expect all students to behave ethically: **do not cheat, plagiarize, or commit fraud**. Fraud includes faking program transcripts to make it appear that code works correctly when it does not; plagiarism includes using someone else's work.

I *highly recommend* the following (inexpensive!) book to help you successfully navigate the academic-honesty waters during your collegiate career:

Charles Lipson, *Doing Honest Work in College: How to Prepare Citations, Avoid Plagiarism, and Achieve Real Academic Success*, The University of Chicago Press, Chicago, Illinois, 2004.

This book introduces the three basic tenets of academic honesty, and also provides excellent suggestions for studying for exams, working in groups, writing research papers, and other tasks you'll encounter as a college student.

If I discover any instances of cheating, fraud, or plagiarism, I will give the guilty parties **failing grades for the course** and report the culprits to the director and the dean. If you are unsure whether a certain action constitutes cheating, fraud, or plagiarism, assume that it does: you may ask us for clarification at any time.

Every student must read and sign a copy of the course **Honor Policy**, which details your obligations to behave ethically. Students will receive zeroes on all assignments/exams until this sheet is turned in to us.

Accommodations

If you need an accommodation because of a disability, then contact both the instructor and the Office of Disability Services (Room 309 in 804 University Ave, phone: x 3-4498). Please do not be shy about this.