

## Course Syllabus

JANUARY 17, 2012

### Catalog Description

Environments, stores, scoping, functional and imperative languages, modules, classes, data encapsulation, types, and polymorphism. Implementation of these constructs in a definitional interpreter. Three hours of lectures. One hour of computer laboratory.

### Instructor

**Jim Royer** (jsroyer at syr dot edu)  
Office: CST 4-185, x1028  
Office hours: TBA & by appointment.

### Teaching Assistant

Yiou Xiao (yixiao at syr dot edu)  
Office: TBA  
Office hours: TBA & by appointment.

### Course webpage

<http://www.cis.syr.edu/courses/cis352>

### Texts

- \* *Basics of Compiler Design*, by Torben Mogensen.
- \* *Learn You a Haskell for Great Good* by Miran Lipovaca, No Starch Press, 2011.

### References

Other notes may be distributed during the semester as necessary.

### Prerequisites

In this course, we will be building on foundations established in CIS 252, CIS 275, and CIS 351.

In CIS 252, you used the functional-programming language Haskell to learn concepts such as recursion, iteration, modularity, and data abstraction, as well as higher-order functions and pattern matching. In this course, we will be using the functional-programming language Haskell. We will spend the first few weeks of the semester getting familiar with Haskell. However, I am explicitly assuming that you've seen the major ideas of functional programming before: if you do not understand the concepts as I review them, it is **your responsibility** to ask for help and to get up to speed.

In CIS 275, you covered basic set theory, functions, predicate logic, and induction. In this course, we will make use of these topics in multiple ways. An important feature of programs written in functional languages is that they are typically much easier to reason about and prove correct than those written in imperative languages. We will make significant use of inductively defined data types and data structures: understanding induction is essential for understanding these beasts and for correctly implementing algorithms that operate on them.

In CIS 351, you saw many data structures—including stacks, queues, lists, and trees—as well as algorithms that operate on them. In this course, we will frequently use trees as internal representations of program code (these representations are known as “abstract syntax”) that we

then manipulate in various ways. We will make use of stacks (or related structures) to implement concepts such as *program environments* or stack frames.

## Objectives

Whenever you learn a new programming language, you're faced not only with learning a new syntax but also with determining what features the language provides, how those features interact with one another, and how to use those features effectively. Furthermore, many programming applications ultimately require you to develop and implement your own small (or not so small) language by which the end user interacts with the application.

This course explores concepts underlying the definition, implementation, and use of programming languages. The goal is to provide you with an understanding of (and a vocabulary for) common language features, including how they are implemented, how other language-design choices affect them, and how they can be used effectively in program development.

This year's edition of the course is an experiment in incorporating elements of computational semantics of natural languages (e.g., English) into our work on programming languages. The studies of natural and programming languages have many elements in common. We (students *and* teacher) will discover how well these mesh as the semester progresses.

We will be using the programming language Haskell.

## Outcomes (Provisional and likely to be revised)

After completion of the course, you should be able to:

- When given an informal but fairly precise English description, write a Haskell program that accurately captures the desired behavior.
- Write data-directed Haskell programs over lists, trees, and other inductively defined data structures.
- When given a moderate-sized Haskell program and relevant input, calculate the result of that program.
- When given a BNF description of concrete syntax and a specific piece of concrete syntax, draw the corresponding abstract-syntax tree.
- When given a Haskell or  $\lambda$ -calculus expression, identify the free variables, identify the bound variables, and calculate the lexical addresses of the bound variables.
- When given a  $\lambda$ -calculus expression, identify all  $\beta$ -redexes and  $\beta$ -reduce the expression.
- When given a small piece of code, calculate its value under a variety of execution scenarios (dynamic or lexical scope, different parameter-passing mechanisms, etc).
- Implement an interpreter for a simple language incorporating many of the constructs listed in the "course topics" section below.

## Measurement

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

### Homework assignments (45% of final grade)

There will be a homework assignment approximately every week: I will drop the lowest homework grade 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.

### Exams (55% of final grade)

There will be five or six in-class quizzes during the semester. The lowest quiz score will be dropped at the end of the semester. There will also be a two-hour *optional* final exam: the exam portion of your final grade will be the greater of (1) your cumulative average of in-class exams, 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 will be made available** on the class web site). Assignments are due by the date and time specified on them: ***No late assignments will be accepted.***

### Topics

Haskell basics. Data and type abstraction.  $\lambda$ -calculus and substitution. Abstract and concrete syntax. Lexical and dynamic scope. Side effects and state. Environments and closures. Recursion. Continuations. Implementation of these constructs in a series of interpreters.

**Academic Integrity** All students should be familiar with Syracuse University's Academic Integrity Policy, which is available at <http://academicintegrity.syr.edu>. 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 without proper credit.

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 me 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 me.

### Disability Accommodations

Students who may need accommodations because of a disability must register with the Office of Disability Services (ODS), 304 University Avenue, Room 309, 315-443-4498. Students with authorized disability-related accommodations should provide a current Accommodation Authorization Letter from ODS to the instructor and review those accommodations with the instructor. Accommodations, such as exam administration, are not provided retroactively; therefore, planning for accommodations as early as possible is necessary. For further information, see the **Office of Disability Services website**.