Overview/important dates

For CIS 194 you will complete a final project which will tie together some of the things you have learned and give you some practical Haskell development experience. The expectation is for you to spend around **15-20 hours** working on the project. Here are some important dates:

- **Wednesday, April 1** – Project proposals due
- **Wednesday, April 29** – Last day of class, final submission due
- **???** – Demo day!

Get started early!

Format

You may work by yourself, or in pairs. Note, however, that projects for pairs will be held to somewhat higher standards than those for individuals.

There are two types of projects you may complete:

1. Application/library

   For your project you may write some sort of Haskell application or library which does something fun/useful/interesting. Your imagination is the limit. Some possibilities/suggestions include:

   - A program to play a game (like tic-tac-toe, Connect 4, othello, gomoku, poker, mancala, …) against the user.
   - A program to solve puzzles like sudoku or kenken.
   - A program to generate random mazes and let the user interactively solve them, or to solve mazes input by the user.
   - An implementation of some interesting data structure like red-black trees, 2-3-4 trees, binomial heaps, or Fibonacci heaps.
   - An parser and interpreter for a small programming language, such as a while language.
   - A raytracer.
   - Take an interesting program you have written in some other language, and figure out how to port/re-implement it in idiomatic Haskell.
   - Write an alternative to a common command-line tool in Haskell; something simple like grep or netcat is OK. For the intrepid, consider implementing an ssh client, HTTP daemon, or similar.
   - Do something cool with APIs available on the Internet. (A Twitter bot, perhaps?)
   - Write a serverside library in Haskell to streamline pragmatic access to some dataset over an HTTP API.
   - Find an existing web service who does not have an official Haskell wrapper and write one yourself. A good example that came up recently in a personal project is **Lob**.
   - Write a toolkit to normalize/transform data: this toolkit would include, for an example, data structures/functions to take a messily-formatted phone number and normalize it to **E.164**, data structures/functions to take a human name and store it in a standard format (**caveat**), data

structures/functions to normalize US postal addresses, etc. This could be massively helpful to other developers who use Haskell for scraping and data processing.

- Build a parser and theorem prover for intuitionistic propositional logic.
- Build a parser, type checker, and interpreter for the lambda calculus.
- Whatever else your creativity suggests!

2. Open-source contribution

   For your project you may choose an open-source library or application on **Hackage** to contribute to. Contributions may include bug fixes, new features, and/or documentation. You are free to work on any project you like, as long as you can find a reasonable way to contribute.

   Open-source projects students have contributed to in prior years include a package to efficiently compute prime numbers using a mutable-array-based sieve and Haskell bindings to the Kinect.

Project proposal

You must submit a project proposal by **Wednesday, April 1**. This gives us a chance to discuss your proposal and ensure it will make a suitable project.

Proposal submission will be on Canvas. Your proposal should answer the questions: What do you propose to do? What do you hope to learn from the project? What are some concrete goals, i.e. how will we judge the success of your project?

There is no formal formatting requirement for the proposal, but I will ask you to revise proposals that are too vague. I should have a decent idea of what the final product will look like, so that way, the TAs and I can evaluate if you achieve what you set out to. Your proposal must explicitly discuss how you plan on testing your work (with QuickCheck).

Final submission

Final submissions are due by **Wednesday, April 29**.

Your final submission should consist of any and all code you have written, along with a document describing your project (a simple text file is fine). The document should contain

- a description of your project and what you accomplished;
- instructions on how to compile/run/try out/play with your project;
- a description of work you did and things you learned along the way.

Submit your project as a compressed file (`.tar.gz`, `.zip`, etc.) through Canvas. If you contributed to an external project, then your submission should contain a specific listing of what, exactly, were your contributions. The code itself can be on, e.g., GitHub – you don't have to submit a copy.

Grading will be as follows:

- Proposal (5%). Did you submit a well-thought out, sufficiently detailed proposal?
- Style (25%). Your project should use good Haskell style and be well-documented.
- Correctness (30%). Your project should be free of compilation errors and should correctly accomplish whatever it is supposed to accomplish. This means that if the deadline is looming, your time would be better spent fixing bugs in what you already have than adding one last feature.
- Demo (15%). Your demo should show off what you have accomplished, what is cool/interesting/novel about your project, and highlight a piece of code that was challenging to write.
- Effort/accomplishment (25%). We will be looking for evidence that you put energy and effort (~15-20 hours) into your project and that you have learned something. This is where the document you submit along with your project comes in: be sure to use it to highlight work you did and things you learned, especially if it is not obvious from looking at the final product. For example, if you spent two hours trying

an approach that ultimately did not work, you should write about that and what you learned from the experience. However, we will not necessarily look with sympathy on *unnecessary* work: for example, if you spent five hours trying to track down a bug without asking for help, that's just plain silly stubbornness. If you are stuck on something, please ask for help. We want you to spend your time making progress on your project, not banging your head against a wall (although a small amount of head-banging can be healthy).

Project demos

A project demo should show off your hard work, show us what's interesting about your project, and highlight a particularly challenging bit of code. I do not expect PowerPoint slides!

The demo day will be most likely be some time during reading days and is a combined demo with students from the other CIS19x courses. Lunch will be served. The demos will be presented in a "science fair" format, with folks wandering around from presenter to presenter. Come with a laptop and show off your work.

The demo is a *required* part of the final project.