The Final Project

Ken Pu, PhD
Computer Science, Faculty of Science, UOIT

Introduction

In place of a final examination, *Programming Languages* opts to have a final project in order to foster individual creativity. Given the class size, we can afford to have individual based projects, so we will not permit any group based projects.

The objective of the final project is to allow students to explore the complex landscape of programming languages based on the principles we have covered in the course. The best way to explore is to select one or more programming languages and use them to solve a specific problem. In the end, you will be evaluated based on several metrics as given in this document.

Selection of language & problems

I would like to make some suggestions on your choice of the programming language and problem to solve. Keep in mind that these are simply suggestions, and you are free to pick something quite different. The real value we want to create is a *learning experience*.

I suggest that you pick *one* language and *one* problem to start. Remember, you will be rewarded for functional and complete programs, so make sure you can achieve executable demo code.

You do not have to select the languages given, as there are dozens and dozens of choices that are equally valid, but I can't really include everything. (For one thing, I don't know all the languages out there). Nor do you need to pick the problem described here. Some languages are highly suitable for the Web (e.g. Google's Dart), while others are highly suitable for numerical and scientific computation (e.g. MIT's Julia).

Evaluation Scheme

1. Github.
   Your code just be in a public repository on *github.com*.
2. Activities
   You must make frequent updates to the *git* repo. This will be verified using "git log" command. You must have an entry to your git repo at least every two days.
3. Problem statement and language selection.
   You must clearly state the problem you are addressing, and the language of your selection to solve the problem.
4. A brief survey of alternatives.
   State how the same problem can be solved by some other languages. You don't have to implement a solution using a second language, but you are expected to know about them.
5. Build tools.
   Describe the build tools required to develop the solution in your language. You should have a *Makefile* whenever possible in your git repo.
6. Describe by means of *code walk* the language features used in your solution. You are expected to highlight the most impactful usage of some language features.
7. Relating to the course.
   Continuing with (6), you should refer back to the topics covered in the course including *type systems, type inference, lexical scoping and closure, functions as data, coroutines, list comprehension*, etc. You can also relate back to Scala, Clojure, Java whenever appropriate to illustrate the similarity and distinction of the language of your choice.
8. Live demo

Evaluation Method

1. The evaluation is conducted in an oral interview format.
2. You are expected to bring a computer with you.
3. You will be asked questions on topics of 1 -- 8 as listed in the *Evaluation Scheme* section.
4. You are *required* to prepare a set of presentation slides to assist your answers. This means that you should address each of the topics (1 - 8) using some slides.
5. You are *required* to submit your presentation using one of the following methods:
   a. As a PDF submitted via *blackboard.com*
   b. As the README.md file that accompanies the github repo.
   c. A publicly accessible Google Slide document.

Appendix:

Language	Features	Suitable types of projects
Scala	We covered enough Scala for you to go deeper into the language features. Scala is *great* at building large scale software. It's *software engineering* features such as Traits, support of functional programming make it an ideal for building commercial systems.	1. Web development using the framework using *Play*. 2. Building a distributed system using *Akka*. 3. Put together an awesome application using these awesome Scala libraries: https://github.com/lauris/awesome-scala
Clojure	We have covered this languages in great depth as well. Now, you get a chance to use it at scale. Clojure is extremely well designed as a swiss army knife for problem solving. It's *software transactional memory* (STM) and concurrency models make programming parallelism a breeze.	1. Web development on the server-side using Clojure web frameworks such as *Compojure*. 2. Experiment with transactions and different STM artifacts including refs, atoms and agents. Understand the trade-off between safety and performance when working with STM. 3. Build an awesome application using these great libraries: http://www.clojure-toolbox.com/ 4. Explore in-browser programming using Clojurescript.
Go	Go, the Google programming language, has gained a great deal of attention due to its small core language, and the awesome first-class support for concurrent sequential processes (CSP).	1. Web development using the server-side Go. It's standard *http* library is great at building high-performance Web services. 2. Build a distributed system using the language features such as goroutines and channels.
Rust	Rust, backed by Mozilla, is becoming a mature language for system programming. It's emphasis is on *safety*. Its syntax is highly expressive for specifying *types*.	1. Learn the language, and implement several classically understood algorithms such as shortest-path, spanning trees, sorting. Compare the performance and programming experience of Rust with other languages.
Haskell	Haskell is definitely a language that stands out. It's a language for the mathematically inclined minds. The source code in Haskell often reads like mathematical proofs; and that's a good thing.	1. Learn the language. Write some classically understood algorithms. Compare the programming experience and performance with some other standard languages.
Erlang	With all the fancy talk of concurrency by the newcomers like Go, Clojure, Scala, ..., it turns out that no one can beat the robustness and reliability of implementing distributed systems using Erlang. Some of the most deployed concurrent systems are written in Erlang: just lookup ejabberd.	1. Learn the language. Understand why it's used for concurrent and distributed systems. Implement a distributed system using Erlang.