COMP4095
Real-world Functional Programming Project
Autumn, Academic Year 2020/21

Henrik Nilsson
School of Computer Science
University of Nottingham

November 21, 2020

1 Introduction

COMP4095 is the optional 10 credit project for the module COMP4075 Real-world Functional Programming. The project is on a topic of your own choice related to real-world functional programming theme of COMP4075, interpreted broadly. It is to be carried out *individually*.

There are two stages to the project:

- *Pitch*: to define the project and get it approved by the module convener
- *Main project*

The deadlines are given on the COMP4075 module webpage.

2 Project Topic and Pitch

The choice of topic is up to each student, and there is no problem if more than one student chose similar projects. The topic can be anything related to what has been covered in the module or broadly within the scope of the module as set out in the module syllabus, but it should involve a programming task. Specifically, there is no requirement that Haskell is used. Indeed, the language used would not even necessarily have to be a "functional" one. What is important is that the ideas underpinning the project is related to real-world

functional programming. And it is up to each student to explain that their proposed topic meets this criterion in their project pitch.

The following is a non-exhaustive list of possible project topics or starting points for defining a topic:

- Using an industrial-strength, functional or functionally inspired framework or library to write a small (demonstration) application. Possibilities include distributed programming, web programming, reactive programming, database interaction, GUI toolkits. A few ideas: MapReduce, ReactiveX, Haxl, FGL (Functional Graph Library).

- Explore a (mostly) functional language other than Haskell to solve some problem or developing some small (demonstration) application. Possibilities include Erlang, OCaml, Idris, Scheme, Racket, Clojure, Scala, F#, PureScript. Or even a functional logic one like Curry or Mercury.

- Explore some specific aspect of the module (or within the scope of the module) in more depth with a view to relevance and utility for practical applications. Could be some specific programming techniques or patterns (e.g. laziness for dynamic programming, monad transformers, alternatives to monad transformers, lenses, zippers, type-level programming), graphs and pure functional programming, combinator parsers (Parsec, UU-Parsinglib, ...) (vs. parser generators), pure functional data structures (vs. traditional ones), testing techniques, proving properties about programs (LiquidHaskell, Idris, Agda), profiling and performance improvement.

- Revisit some suitable piece of coursework from another module and solve it in a more functional way.

- Implement a simple game. Could be a video game, turn-based game, puzzle solving single-player game (or generator of such puzzles). Might involve multiple players over the network.

As a first step, the chosen project needs to be pitched to the module convener for approval. The pitch should be in the form of a brief, plain-text e-mail directly to the module convener. The pitch must:

- outline what the project is about

- state what language(s), framework(s), etc. are going to be used

- explain why the proposed project is relevant for the module

- give a rough time estimate for the project: in all, including project definition, it should take about 100 hours (as this is a 10-credit module)
- state what the outcome of the successful project is

The approval process will be very light-weight, but if necessary, the pitch will have to be amended until the above criteria are met.

3 Project Assessment and Submission

The following has to be submitted by the deadline:

- A brief written report as specified below.
- The complete source code of what has been developed.

The submission is electronic only:

- Electronic copy of the report (PDF). The file should be called `psyxyz-report.pdf`, where `psyxyz` should be replaced by your University of Nottingham user ID.
- Archive of the source code hierarchy (gzipped TAR, or zip). The archive should be called `psyxyz-src.tgz` or `psyxyz-src.zip`, where `psyxyz` again should be replaced by your University of Nottingham user ID, and it should contain a single top-level directory containing all the other files.

The report would typically cover the following:

- Overview of the project.
- Brief technical background to make the report reasonably self-contained.
- Discussion of the implementation, justifying key decisions and highlighting and explaining particularly interesting aspects, illustrating with excerpts from the developed code where appropriate. Report on testing.
- A section reflecting upon what was learned from the project and your thoughts around the project topic from a real-world programming perspective.

This outline may have to be adapted depending on the exact nature of the project. As a guide, the report should be around 10 pages or 5000 words long, excluding large code fragments, pictures, and any appendices. Sources must be properly referenced.

The submitted source code should be accompanied by a plain text README file in the root directory giving a brief overview of the source code hierarchy and making clear which code was written from scratch, which code has been adapted (e.g. from tutorials), and which code is third-party code.

The work will be assessed on the difficulty of what was attempted, the extent to which the project was successful, quality of the code as well as of the report. In more detail, the following aspects are considered,each assessed on a scale from 0 to 10 and weighted as indicated:

- Challenge, 30 %: Difficulty of what was actually achieved
  - Max: 10
  - Reference:

    9: Difficult project substantially meeting its aims, or standard project substantially exceeding its aims

    7: Project of average difficulty substantially meeting its aims

    5: Very easy project, or relatively little actually achieved

    2: Very easy project, yet very little achieved

- Code, 30 %: Aspects, where applicable given the nature of the project, such as: use of appropriate (functional) abstractions; appropriate use of types; organisation; systematic and helpful naming conventions; formatting; documentation (comments); testing
  - Max: 10
  - Reference:

    10: Impeccable

    7: Very good, but still room for improvement.

    5: No major flaws, but substantial room for improvement

    1: Very poor in most respects

- Report, 30 %: Quality of the report as a technical document: overall organisation, writing, typesetting, figures, citations
  - Max: 10
  - Reference:

10: Impeccable

7: Very good, but still room for improvement.

5: No major flaws, but substantial room for improvement

1: Very poor in most respects

- Reflection, 10 %:
  - Max: 10
  - Reference:

    10: Highly insightful with a number of really useful lessons clearly and compellingly argued

    6: Moderately insightful reflection with some appropriate lessons clearly articulated

    4: OK, but some questionable conclusions put forward without adequate support

    1: Very Little of substance put forward; little if any support for expressed opinions.