S0.1: Course Introduction

CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2019

prin·ci·ple

noun: principle; plural noun: principles

1. a fundamental truth or proposition that serves as the foundation for a system of belief or behavior or for a chain of reasoning.

What is our "system of belief or behavior" that our principles will support?

Beliefs about programming

We might ask what makes a piece of software "good"?

What properties does "good" software have?

-
-
-
-

What principles support these goals?

Correctness

- Principle:
  - Strong static types.

- Principle:
  - Proofs that code meets some correctness specification.
  - Inductive proofs based on equational reasoning.

"Program testing can be used to show the presence of bugs, but never to show their absence!"

— Edsger Dijkstra

"Be careful about using the following code – I've only proven that it works, I haven't tested it."

— Donald Knuth

Efficiency

- Principle:
  Without correctness, efficiency is meaningless.
- Principle:
  Use algorithms with acceptable computational complexity.
- Principle:
  Programs should effectively and efficiently use all available resources.

"Rules of Optimization:

Rule 1: Don't do it.

Rule 2 (for experts only): Don't do it yet."

— M.A. Jackson

"More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity."

— W.A. Wulf

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil."

— Donald Knuth

Re-usability

- Principle:
  Procedural/functional and data abstraction
- Principle:
  Decomposition into modules
- Principle:
  Extensiblity

"Transparency"

- Principle:
  We need to be able to read and understand software.
- Principle:
  Our compiler needs to be able to reason about programs to detect errors and perform optimizations.
- Principle:
  Work with complex, symbolic, inductive data.
  Focus on abstract view, not machine representation.

Software needs to be "transparent", and not "opaque."

"Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."

— Donald Knuth

Varieties of computation

In considering these principles, we will study different styles of computation:

Computation as (effect-free) expression evaluation

- Functions as values.
- Various evaluation mechanisms
  - eager
    ```
    map square [1; 2+3; 4+5+6; 7+8+9+10]
    ```
  - lazy
    ```
    take 5 (map square (allIntsFrom 10))
    ```
  - parallel
    ```
    map square (mkList 1 1000)
    ```

Effectful computations

- traditional imperative programming
- understand this as state transformation
- “denotational semantics”

Value-based programming

- immutable data
- immutability in impertative programming
- challenges to efficiency
  - Okasaki: Purely Functional Data Structures

Search as a computational style

- computation as search for a solution

Concurrency

- a style of programming
- a way to organize programs
- concurrency $\neq$ parallelism

Why a functional language? Why OCaml?

Many of these principles and techniques are more directly expressed in modern statically-typed functional languages.

- "Fine, but is this all *academic*?"
- "How useful is any of this in the real world?"

2 answers:

1. Who cares!
2. Very useful. (CPS in AJAX, Scala/Java 8, etc.)

Using OCaml

- Some, however, claim that OCaml itself, not just the ideas embodied in it, is a useful "real world" programming language.
- To understand this perspective, read "OCaml for the Masses" by Yaron Minsky.

  It is on GitHub, in `Resources`.

Course logistics

- Below are some syllabus excerpts.
- Read the syllabus!
- I'm serious. The syllabus contains the rules and policies for the course. Not knowing these is foolish.
- We only discuss a few of them here.

Additional Policies

All policies (presented above) may evolve and change over the course of the semester at the discretion of the instructor. Sometimes issues arise that cannot (or were not) planned for, and I may need some flexibility in handling them.