S0.1: Course Introduction

CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2022

Welcome!

- Welcome to CSCI 2041: Advanced Programming Principles.
- Call me "Eric", "Dr. Van Wyk", or "Professor Van Wyk" as you prefer
- pronouns: he/him/his
- Everything is at `https://z.umn.edu/csci-2041-10-S22`

Welcome, and oh drat!

- Well, Covid-19 is still a thing.
- Thus, several of the remote teaching mechanisms from last year will still be with us.
  - live-streamed and recorded lectures
  - lab attendance is optional
  - office hours are online

  All aimed at letting you stay home and also making a less densely populated classroom.
- In fact, I strongly encourage you to stay away for the next few weeks.
- More flexibility with due dates, as we'll see.
- If you have concerns, please email me and we can discuss it.

Masks

- Masks are required - but get good ones!
- I'll always have a few in case you're without one, just ask.
- I do recommend getting N95s if you can. They are readily available these days.
- The U is giving out better masks now - see the syllabus for where to get them.
- This does preclude eating and drinking in class.
- We'll figure it out.

What is "Advanced Programming Principles" about?

- What do we make of this word "Principles" in the course title?

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 system of beliefs or behaviors do our principles support?

Beliefs about programming

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

What properties does "good" software have?

-
-
-
-

These properties form our "system of belief."

What principles support these?

Correctness

Principle: Strong static typing.

- these impose a certain discipline when writing programs
- they avoid entire classes of erroneous behaviors

Principle: Prove 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.

"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: In working 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.
- Use of effectful computations.
- Value-based programming.
- Search as a computational style.
- Concurrency.

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 imperative programming
- challenges to efficiency
  - Okasaki: Purely Functional Data Structures

Search as a computational style

- computation as a search for a solution
- what is the "search space"?
- how can it be explored?

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. You may find it to be quite convenient and productive.

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.

  On the Course-Resources page of the course web page.

Course logistics

- We'll take a look at 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 Comments

Lecture Format

- Lectures will present material not in the texts.
- Lecture slides, in various formats, will be available on the course web page.
- But, many most lectures will have a significant live programming and whiteboard-like components that will not be found in the slides.
  - Be prepared to take notes.
  - Slides online are in "3up" format. There is room for notes on those.
- Several small in-class exercises will be done during lectures.
  So come prepared to work.
- We'll discuss how to submit these on Friday.

How get answers to your questions?

I **want** to talk to all of you, but...

We ask you follow these in order ...

1. Raise your hand!
2. See a TA.
3. Ask a question on Piazza.
   - to everyone in the class
   - to "instructors"
   - to me

Laptops: Lectures are interactive

- Please close your laptop.
- Laptops are a distraction to you and to those seated behind you and thus are **not allowed** to be open during lecture, except for some in-class coding exercises.
- Laptops, physically, get in the way of on-paper exercises and discussion with other students.
- Similarly, no tablets or cell phones.

Starting Friday

- The fun starts!
- Start learning more about OCaml