Programming Languages and Techniques (CIS120)

Lecture 1

Welcome
Introduction to Program Design
Introductions

- **Eric Fouh**
  - Levine Hall 603
  - efouh@cis.upenn.edu
  - http://www.ericfouh.com
  - Office hours: Tuesdays 2:00-4:00pm (and by appointment)

- **Benjamin Pierce**
  - Levine 562
  - http://www.cis.upenn.edu/~bcpierce/
  - bcpierce@cis.upenn.edu
  - (tentative) Office hours: Tuesdays, 1:00 – 3:00 (and by appointment)
What is CIS 120?

• CIS 120 is a course in program design

• Practical skills:
  – ability to write larger (~1000 lines) programs
  – increased independence ("working without a recipe")
  – test-driven development, principled debugging

• Conceptual foundations:
  – common data structures and algorithms
  – several different programming idioms
  – focus on modularity and compositionality
  – derived from first principles throughout

• It will be fun!
Prerequisites

• We assume you can already write small (10- to 100-line) programs in some imperative or object-oriented language
  – Java experience is strongly recommended
  – CIS 110 or AP CS is typical
  – You should be familiar with using a compiler, editing code, and running programs you have created

• CIS 110 is an alternative to this course
  – If you have doubts, come talk to an instructor or one of the TAs to figure out the right course for you
CIS 120 Tools

• OCaml
  – Industrial-strength, statically-typed *functional* programming language
  – Lightweight, approachable setting for learning about program design

• Java
  – Industrial-strength, statically-typed *object-oriented* language
  – Many tools/libraries/resources available

• Eclipse
  – Widely used IDE
• Codio codio.com
  – web-based development environment
  – see Piazza / class mailing list for setup info
  – *remote access for on-line TA help*

• Under the hood:
  – linux virtual machine (Ubuntu)
  – pre-configured per project with everything you need
Why two languages??

- Clean pedagogical progression
- Level playing field for class with disparate backgrounds
- Practice in learning new tools
- Different perspectives on programming

--- Anonymous CIS 120 Student

"[The OCaml part of the class] was very essential to getting fundamental ideas of comp sci across. Without the second language it is easy to fall into routine and syntax lock where you don't really understand the bigger picture."

--- Anonymous CIS 120 Student

"[OCaml] made me better understand features of Java that seemed innate to programming, which were merely abstractions and assumptions that Java made. It made me a better Java programmer."

--- Anonymous CIS 120 Student
Philosophy

• Introductory computer science
  – Start with basic skills of “algorithmic thinking” (AP/110)
  – Develop systematic design and analysis skills in the context of larger and more challenging problems (120)
  – Practice with industrial-strength tools and design processes (120, 121, and beyond)

• Role of CIS120 and program design
  – Start with foundations of programming using the elegant design and precise semantics of the OCaml language
  – Transition (back) to Java after setting up the context needed to understand why Java and OO programming are useful tools
  – Give a taste of the breadth and depth of CS
Administrivia

http://www.seas.upenn.edu/~cis120/
Course Grade Breakdown

• Lectures (2% of final grade)
  – Presentation of ideas and concepts, interactive demos
  – Lecture notes & screencasts available on course website.
  – Grade based on participation using “Poll Everywhere”

• Recitations / Labs (6% of final grade)
  – Practice and discussion in small group setting
  – Grade based on participation

• Homework (50% of final grade)
  – Practice, experience with tools
  – Exposure to broad ideas of computer science
  – Grade based on automated tests + style
  – First assignment due in 14 days – available Friday

• Exams (42% of final grade)
  – 2 midterms (12% each) and a final (18%)
  – In class exams (pencil and paper)
  – Do you understand the terminology? Can you reason about programs? Can you synthesize solutions?

Warning: This is a challenging and time consuming (and hopefully rewarding :-)) course!
Some of the homework assignments...

Computing with DNA

Building a GUI Framework

Image Processing

Chat Client/Server
Final project: Design a Game
Lectures / Recitations / Lab Sections

• The lecture material in the sections 001 and 002 will be identical

• Recitations start next week
  – Bring your laptops to recitation
  – Try Codio before the first meeting

• Goals of first meeting:
  – Meet your TAs and classmates
  – Practice with OCaml before your first homework is due

• Office hours times on the web site calendar (under “Help” tab)
  – Will be filled out soon
Poll Everywhere

• We will use *Poll Everywhere* for interactive exercises during most lectures
  – Wrong answers do not count against your grade

• You can use your phone/laptop to go to a website to post your answer.

• You can also use your phone to text the answer directly

• Bring it to lecture every day, beginning Friday
  – Participation grades start Monday in 2 weeks (in 13 days)
No Devices

• **Laptops closed... minds open**
  – Although this is a computer science class, the use of electronic devices – laptops, cellphones, Kindles, iPads, etc., during lecture (except for participating in quizzes) is *prohibited*.

• **Why?**
  – Laptop users tend to surf/chat/e-mail/game/text/tweet/etc.
  – They also distract those around them
  – Better to take notes *by hand*
  – You will get plenty of time in front of your computers while working on the course projects :-)

CIS120
We will use Piazza for most communications in this course:
- from us to you
- from you to us
- from you to each other

If you are registered for the course, you should have been signed up automatically.

If not, please sign up at piazza.com.
Academic Integrity

• Submitted homework must be *your individual work*

• Not OK:
  – Copying or otherwise looking at someone else’s code
  – Sharing your code in any way
    (copy-paste, github, paper and pencil, ...)
  – Using code from a previous semester

• OK (and encouraged!):
  – Discussions of concepts
  – Discussion of debugging strategies
  – Verbally sharing experience

*Penn’s code of academic integrity:*
http://www.upenn.edu/academicintegrity/ai_codeofacademicintegrity.html
Enforcement

• Course staff will check for copying
  – We use plagiarism detection tools on your code

Violations will be treated seriously!

• Questions? See the course FAQ. If in doubt, ask.
Program Design
**Fundamental Design Process**

*Design* is the process of translating informal specifications (“word problems”) into running code.

1. **Understand** the problem
   What are the relevant concepts and how do they relate?

2. **Formalize the interface**
   How should the program interact with its environment?

3. **Write test cases**
   How does the program behave on typical inputs?
   On unusual ones? On erroneous ones?

4. **Implement** the required behavior
   Often by decomposing the problem into simpler ones and applying the same recipe to each

5. Revise / Refactor / Edit
Imagine that you own a movie theater. The more you charge, the fewer people can afford tickets. In a recent experiment, you determined a relationship between the price of a ticket and average attendance. At a price of $5.00 per ticket, 120 people attend a performance. Decreasing the price by a dime ($0.10) increases attendance by 15. However, increased attendance also comes at increased cost; each attendee costs four cents ($0.04). Every performance also has a base cost of $180. At what price do you make the highest profit?
Step 1: Understand the problem

• What are the relevant concepts?
  – (ticket) price
  – attendees
  – revenue
  – cost
  – profit

• What are the relationships among them?
  – profit = revenue – cost
  – revenue = price * attendees
  – cost = $180 + attendees * $0.04
  – attendees = some function of the ticket price

• Goal is to determine profit, given the ticket price
Step 2: Formalize the Interface

Idea: we’ll represent money in cents, using integers*

comment documents
the design decision

(* Money is represented in cents.
*)

let profit (price : int) : int = ...

* Floating point is generally a bad choice for representing money: bankers use different rounding conventions than the IEEE floating point standard, and floating point arithmetic isn’t as exact as you might like. Try calculating 0.1 + 0.1 + 0.1 sometime in your favorite programming language...

**OCaml will let you omit these type annotations, but including them is mandatory for CIS120. Using type annotations is good documentation; they also improve the error messages you get from the compiler. When you get a type error message from the compiler, the first thing you should do is check that your type annotations are there and that they are what you expect.

CIS120
Step 3: Write test cases

• By looking at the design problem, we can calculate specific test cases

```haskell
let profit_500 : int =
    let price = 500 in
    let attendees = 120 in
    let revenue = price * attendees in
    let cost = 18000 + 4 * attendees in
    revenue - cost
```
Writing the Test Cases in OCaml

• Record the test cases as assertions in the program:
  – the *command* `run_test` executes a test

  a *test* is just a function that takes no input and returns true if the test succeeds

  ```
  let test () : bool = 
  (profit 500) = profit_500

  ;; run_test "profit at $5.00" test
  ```

  note the use of double semicolons before commands

  the string in quotes identifies
  the test in printed output
  (if it fails)
Step 4: Implement the Behavior

**profit, revenue, and cost** are easy to define:

```plaintext
let attendees (price : int) = ...

let profit (price : int) =
    (revenue price) - (cost price)
```
apply the design pattern recursively

**attendees** requires a bit of thought:

```plaintext
let attendees (price : int) : int =
    failwith "unimplemented"

let test () : bool =
    (attendees 500) = 120
;; run_test "attendees at $5.00"
```  

```plaintext
test

let test () : bool =
    (attendees 490) = 135
;; run_test "attendees at $4.90"
```  

*Note that the definition of attendees must go before the definition of profit because profit uses the attendees function.

“stub out” unimplemented functions

generate the tests from the problem statement first.
Assume a linear relationship between ticket price and number of attendees.

Equation for a line: \( y = mx + b \)

- \( m \) = (diff in attendance / diff in price) = -15/10
- \( b \) = attendees – \( m \) * price = 870

Let attendees (price:int) : int =

\[-15/10 \times \text{price} + 870\]
Run!
Run the program!

- One of our test cases for attendees failed...
- Debugging reveals that integer division is tricky*
- Here is the fixed version:

```plaintext
let attendees (price:int) : int =
(-15 * price) / 10 + 870
```

*Using integer arithmetic, \(-15 / 10\) evaluates to \(-1\), since \(-1.5\) rounds to \(-1\). Multiplying \(-15 * \text{price}\) before dividing by 10 increases the precision because rounding errors don’t creep in.
Modern approaches to software engineering advocate test-driven development, where tests are written very early in the programming process and used to drive the rest of the process.

We are big believers in this philosophy, and we’ll be using it throughout the course.

In the homework template, we may provide one or more tests for each of the problems. They will generally not be complete. You should start each problem by making up more tests.
How not to Solve this Problem

```haskell
let profit price =
    price * (-15 * price / 10 + 870) -
    (18000 + 4 * (-15 * price / 10 + 870))
```

This program is bad because it

- hides the structure and abstractions of the problem
- duplicates code that could be shared
- doesn’t document the interface via types and comments

*Note that it still passes all the tests!*
Summary

• **To read:** Chapter 1 of the lecture notes and course syllabus. Both available on the course website.

• **To do:** Sign up for Codio and try to log in.
  – TAs will hold office hours this week to help.
  – You can also use Piazza for discussions.

• **To do:** Register for Poll Everywhere.
  – Polls will start on Friday.
Textbook

- Textbook (free download)
  - [http://www.seas.upenn.edu/~cis120/current/notes/120notes.pdf](http://www.seas.upenn.edu/~cis120/current/notes/120notes.pdf)
  - written by the course instructors, closely follows the lectures
  - updated throughout the semester