Slide 1

Administrivia

- One purpose of the syllabus is to spell out policies (next slides).
- Most other information will be on the Web, either on my home page (here, office hours) or the course Web page (here).

  A request: If you spot something wrong with course material on the Web, please let me know!

Slide 2

Course FAQ

- "What will my grade be based on?" (See syllabus.)
- "What happens if I can't turn in work on time, or I miss a class?" (See syllabus.)
- "What's your policy on collaboration?" (See syllabus.)

Slide 3

Course FAQ, Continued

- "When is the next homework due?" (See "Lecture topics and assignments" page.)
- "When are your office hours?" (See my home page.)

  Note that part of my job is to answer your questions outside class, so if you need help, please ask! in person or by e-mail or phone.

Slide 4

Course FAQ, Continued

- "What computer(s) can I use to do homework?"

  Easiest option may be department's Linux lab machines. There are others.

  You should have physical access (via your TigerCard) to five rooms containing such machines any time the building is open. You should have remote access to any that are booted into Linux.

  Returning students should already have accounts set up. (If you've forgotten your password, go to the ITS help desk and ask for it to be reset.) To change your password, open a terminal window and type `passwd`.

Slide 5

What Is This Course About?

- Back story: Primary goal of our traditional first course (CSCI 1320) is to introduce students to programming and algorithmic problem-solving. Another goal of the course as taught up to this year, however, was to expose students to certain low-level concepts that contribute to a well-rounded education in computer science. Students coming into the major via other routes often did not get this exposure and struggled in later courses.
- CSCI 1120 was added to the curriculum as a way to address this problem — i.e. to cover the parts of CSCI 1320 that might not be covered by alternative introductory courses. With the recent shift in language(s) used in CSCI 1320, it will be required for all students.

Slide 6

Course Topics

- Basic C programming, for people who already know how to write programs in some other language.
- (Review of) the Linux/UNIX command-line environment and command-line development tools.
- (Review of) basics of computer arithmetic.
- More advanced topics as time permits.

Slide 7

Programming Basics (as described in CSCI 1320)

- What computers actually execute is *machine language* — binary numbers each representing one primitive operation. Once upon a time, people programmed by writing machine language (!).
- Nowadays, "programming" as we will use it means writing *source code* in a *high-level language*. Source code is simply plain text, which . . . At this point we diverge from the explanation for beginners. Exactly what happens to get from source code to something the computer can execute varies among languages . . .

Slide 8

From Source Code to — What?

- Some high-level languages (such as the language understood by typical UNIX/Linux command shells) are directly interpreted by some other program.
- Others are *compiled* into *object code* (machine language) and then *linked* with other object code (including system libraries) to form an *executable* (something the operating system can execute).
- Still others (including Scala and Python, sometimes) take an intermediate approach — initially compiled into *byte code* (object code for a made-up processor), which is (in principle) interpreted by a runtime system, with system library code brought in at runtime. (In practice, a "just-in-time" compiler may translate byte code into native object code on the fly.)

Slide 9

Why Learn C? (For Java/Python/Scala Programmers)

- Scala and Python (and Java, less so) provide a programming environment that's nice in many ways — lots of safety checks, nice features, extensive standard library. But they hide a lot about how hardware actually works.
- C, in contrast, has been called "high-level assembly language" — so it seems primitive in some ways compared to many other languages. What you get (we think!) in return for the annoyances is more understanding of hardware — and if you do low-level work (e.g., operating systems, embedded systems), it may well be in C.

Slide 10

Structure of a C Program

- Pre-processor directives: These begin with # and are used to (among other things) include in the compilation process information about libraries.
- Global identifiers (functions and variables). Function declarations here are often useful; variables are usually bad practice.
- Function(s), possibly containing variables, returning values, etc. Every complete program has exactly one `main` function.
- Syntax should look familiar to Java programmers (no accident — Java was designed that way). Less familiar to Python and Scala programmers.

Slide 11

A First C Program

- Let's write the traditional "hello world" program in C, using `vi`.

  (This tradition of having one's first program in a language print "hello world"? It comes from the early and still fairly authoritative book *The C Programming Language*, by Kernighan and Ritchie.)
- Once it's written, compile-and-link by typing `gcc hello.c`. (There are other options you should use, but for now this is okay.) Result is `a.out`.
- Execute by typing `a.out`.
- (We will look at details next time.)

Slide 12

Minute Essay

- Tell me about your background:
  - What programming classes have you taken (high school or other), and what language(s) did you use?
  - How comfortable are you with the Linux/Unix command-line interface?
- What are your goals for this course? Anything else you want to tell me?
- (Reminder: Reading assignment for next time.)