

Slide 1

Administrivia

- If you didn't get your password sorted out already, ask me after class (time permitting) — ITS prefers that you go through them but I can help too.
- For minute essays, last time we used paper, and that's fine, but I'd rather have e-mail. Please put "minute essay" and the course name in the Subject line. I get a *lot* of e-mail and kind of rely on Subject to help me know what's what! (I'll try to respond to individual essays soon. In general, questions you ask will get answered either in e-mail or in the next class.)
- The ASO person working on TigerCard access to our rooms says "maybe Friday" — ? Rooms that will be available to you are CSI 257, 388, and 488, when not in use for classes.
- Note reading assignments for next week — my notes and Dr. Lewis's video lectures on the Linux command line, and material from the textbook.

Slide 2

Minute Essay From Last Lecture

- Most but not all students are ENGR majors. I'll try to aim at those students without leaving others out.
- ENGR majors have all seen Matlab and some have used VHDL. Only one person has experience with a "real" programming language.
- ENGR majors also have significant math background.
- Almost no one claims much experience with command-line environments or Linux. (So after a little more intro stuff we'll do that first.)

Slide 3

Solving Problems on Computers

- Appearances (maybe?) to the contrary, computers are not smart. What they do well is perform sequences of simple math/logic operations very fast and very accurately.
- What makes them useful is that people have figured out how to break complicated tasks down into sequences of simple operations — i.e., how to “program” them.
- This requires a mindset not quite like that required for any other activity — and can involve a lot of creativity.
- It also involves a form of experimentation (which is why our introductory courses meet the old CC “Using Scientific Methods” requirement).

Slide 4

Steps in Solving Problems on a Computer

- Understand the problem — what do you want the computer to do, exactly? (Not as simple as it might sound!)
- Design a solution suitable for a computer (“develop an algorithm”).
- Implement the solution (“write the program”). Requires expressing your ideas in “a programming language” — and there are many! Programming languages similar to human languages in some ways, different in others; meant to be (somewhat!) human-readable while still being precise enough for a computer to understand.
- Test your solution. Involves the use of some tool that translates what you write (“source code”) into something the computer hardware can work with.

Solving Problems on a Computer, Continued

Slide 5

- Overall process — understand the problem, develop and test a solution — mostly independent of choice of programming language and platform (combination of hardware and operating system, roughly). So once you understand the principles, relatively easy to learn new languages.
- Opinions about which language to learn first, and on what platform, vary. For this course we will use C, mostly at the request of ENGR. Not as easy to use as some other choices, but they think more likely to be useful to their students. Not used a lot in general-purpose application development these days but still in use for “embedded systems” and closer to the hardware. We will also do most work from the command line under Linux.

Programming Basics

Slide 6

- What computers actually execute is *machine language* — binary numbers each representing one primitive operation. Once upon a time, people programmed by writing machine language (!).
- Obviously tedious and error-prone. Very early bright idea was to write something more human-readable (*source code*) and *have the computer translate it*. Useful even if the source code is just a human-readable version of the primitive operations (*assembler language*). Even better if the source code is less primitive (*high-level language*).
- Source code is simply plain text (as opposed to text plus formatting, as in a word-processor document). Since the hardware doesn't understand it, however, ...

Programming Basics, Continued

Slide 7

- Source code can be *interpreted* — translated line by line into something the hardware can understand, by another program called an *interpreter*.
(This is how “scripting languages” work. An example is the command shell’s language. !)
- Or it can be *compiled* — translated by a program called a *compiler* into something the hardware can execute directly.
(This is how traditional “high-level” languages such as C and Fortran work.)
- Or it can be compiled into some intermediate form that can be executed by another program.
(This is how some recent languages such as Java and Scala work.)

Writing Source Code

Slide 8

- How do you get source code? If using an interpreter, you can just type it in. If you want something you can keep and reuse, however, you need a tool that will do that.
- Simplest way to create source code is with a *text editor* — a program for writing and editing plain text. This is what we will do for now.
- (Another way is to use an *IDE* (Interactive Development Environment). We won’t use one of those in this class, but if you work on larger projects and/or in other programming languages you may want to.)

A Word About Tools

Slide 9

- In this class we use Linux and command-line tools. We think it's important that CS majors know something about this environment, and we think it can be useful for people in other STEM disciplines as well — and we hope “new-to-me and different” has some appeal for all!
- (What is Linux? it's an operating system, as Windows and Mac OS X are operating systems. It's one of a family of operating systems descended from UNIX, developed at Bell Labs in the early 1970s. A lot of servers run Linux or some other UNIX-like system. There are also ongoing efforts to develop mainstream desktop systems.)
- A UNIX person's response to claims that UNIX isn't user-friendly: “Sure it is. It's just choosy about its friends.”

Getting Started with Linux

Slide 10

- When you log in, you should get a graphical desktop, which should be navigable with what you know from using other graphical environments (though some details are different).
- In Linux, we talk about files and directories; the idea is the same as Windows' files and folders, though again some details are different.
- The graphical system should give you a way to get a terminal window. Once you have that . . . (to be continued)
- (One command for now — to change your password, type `passwd` and follow the prompts. Note that this command doesn't echo what you type, even with placeholders.)

Slide 11

More About Tools / Administrivia

- One option for homework is one of our classrooms. These machines are dual-boot (Windows 10 and Linux). If you get a Windows login screen, reboot, and you should get a menu that lets you select which O/S to launch.
- Another option is to log in remotely: You can get the equivalent of terminal window by using ITS's VDI interface and PuTTY. I will e-mail details (since these slides are public).

Slide 12

Minute Essay

- Anything today that was particularly unclear?