

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 (!).
- Slide 2
- 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 ...

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).

Slide 3

• 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.)

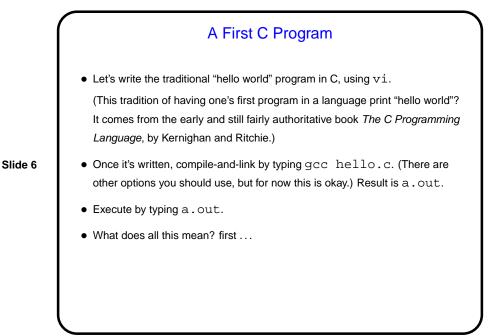
Structure of a C Program

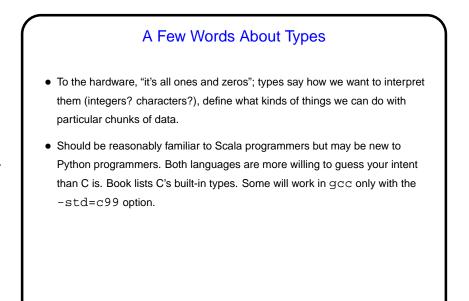
- Pre-processor directives: These begin with # and are used to (among other things) include in the compilation process information about libraries.
- Global identifiers (functions and variables). Function declarations here are often useful; variables are usually bad practice.
- Slide 4
- Function(s), 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.

A Few Words About "Old C" Versus "New C"

- First ANSI standard for C 1989. Widely adopted, but has some annoying limitations.
- Later standard 1999. Many features are widely implemented, but few compilers support the full standard, and older programs (and some programmers concerned about maximum portability) don't use new features. Much of what we do in this class will focus on older standard for this reason.

Slide 5

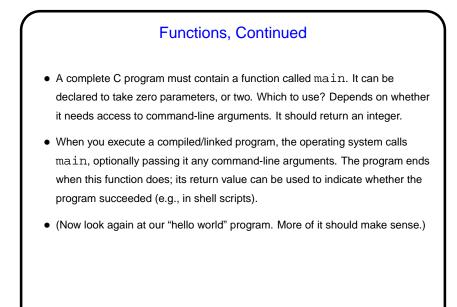


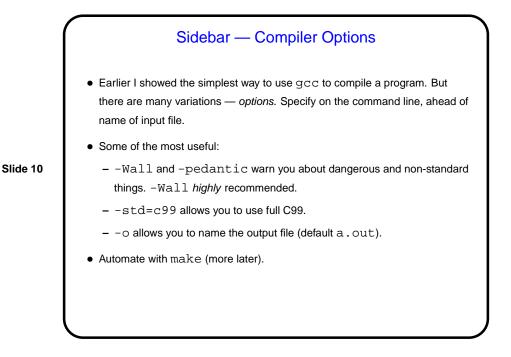


Slide 7

Functions C programs are organized in terms of *functions* — a somewhat more primitive version of methods as found in object-oriented programming languages such as Python and Scala. As in other programming languages, C functions are a little like mathematical functions, except that evaluating them can have "side effects". (For example, evaluating the library function printf has the side effect of writing some text to standard output (by default, displaying it in the terminal window).) Unlike in some other languages, C functions have to be declared (or defined) before being referenced. Declaration includes name, return type, and formal parameters. For library functions, declaration is usually supplied via a #include preprocessor directive.

Slide 8





Slide 9

