

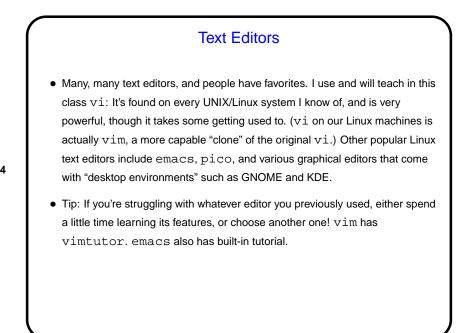
Getting Started with Linux (Review) (A UNIX person's response to claims that UNIX isn't user friendly: "Sure it is. It's just choosy about its friends.") When you log in, you should get a graphical desktop, which should be navigable with what you know from using other graphical environments (though some details are different). The graphical system should give you a way to get a terminal window, which is what we will use a lot in this class (in keeping with the title!). In theory you know the basics from CSCI 1320. If not, review the relevant chapter of the book Dr. Lewis is writing for POP I/II.

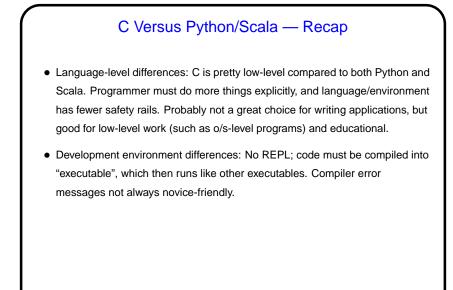
Useful Command-Line Tips

• The shell (the application that's processing what you type) keeps a history of commands you've recently typed. Up and down arrows let you cycle through this history and reuse commands.

(Pedantic aside: "The shell" here means the one you're most likely to be using. There are other programs with similar functionality you could use instead.)

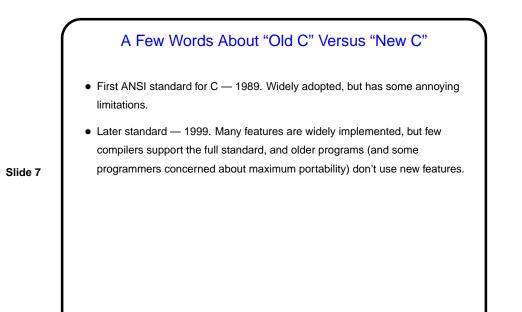
- The shell offers "tab completion" for filenames if you type part of a filename and press the tab key, it will try to complete it.
- To learn more about command foo, type man foo. (This also works with C library routines more about them later.) This is reference information rather than a tutorial, but usually very complete.



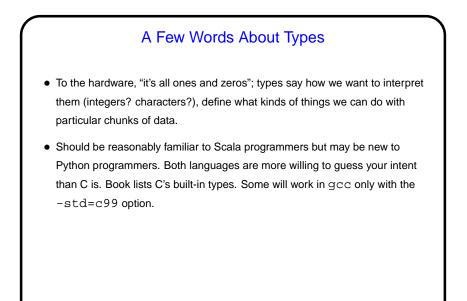


Slide 6

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.



A First C Program, Revisited
Last time we wrote the traditional "hello world" program in C, compiled/linked it (with gcc) and executed it.
Now let's look at the program line by line. But first ...



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

Functions

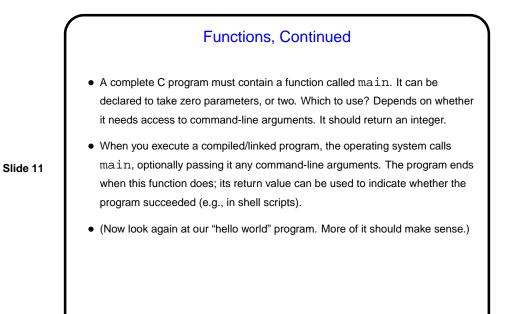
Slide 10

effects".

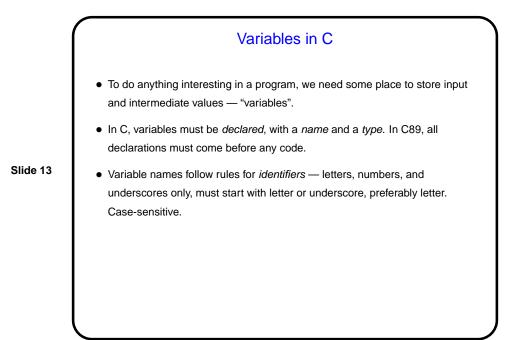
(For example, evaluating the library function printf has the side effect of writing some text to standard output (by default, displaying it in the terminal window).)

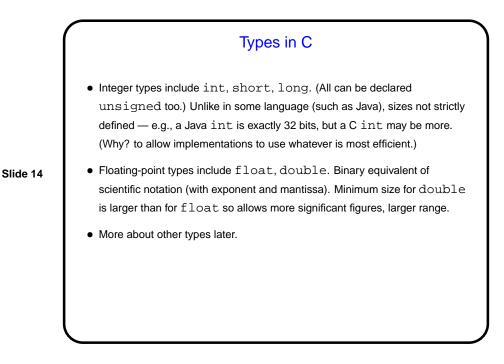
little like mathematical functions, except that evaluating them can have "side

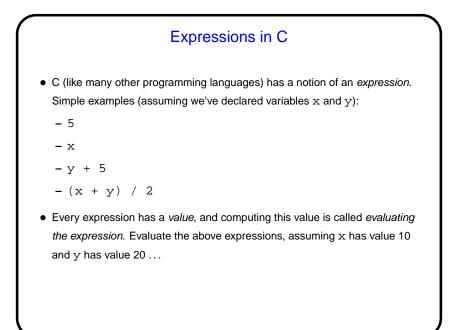
• Unlike in some other languages, C functions have to be declared (or defined) before being referenced. Declaration includes name, return type, and formal parameters. For library functions, declaration is usually supplied via a #include preprocessor directive.

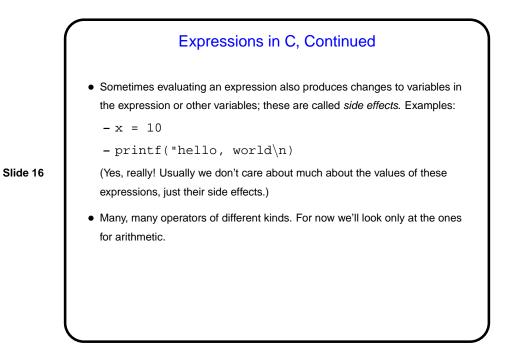


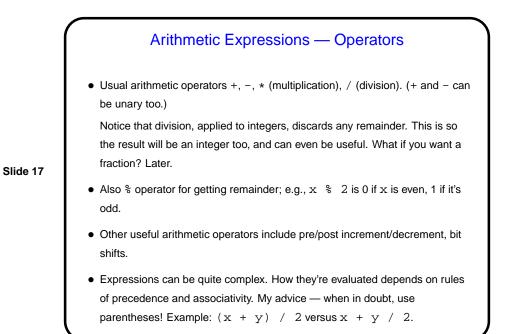
Sidebar — Compiler Options • Earlier I showed the simplest way to use gcc to compile a program. But there are many variations - options. Specify on the command line, ahead of name of input file. • Some of the most useful: Slide 12 - -Wall and -pedantic warn you about dangerous and non-standard things. -Wall highly recommended. - -std=c99 allows you to use full C99. - -o allows you to name the output file (default a.out). • Automate with make (more later).











Statements in C

• C programs are made up of statements (usually collected inside functions.
• Statements come in several types:
 - Null (;).
 - Expression (expression ;).
 - Return (return expression ;).
 - Compound (more later).

Output

- The "hello world" used printf to print some text. printf can do a lot more.
- For example, we can use it to print integers, e.g.,

```
printf("the value of x is dn", x);
```

