Administrivia

Slide 1

- Code examples from class will show up on the "Sample programs" page sometime after class, as soon as I can put them there (but sometimes the next day).
- Do your TigerCards give you access to this room and CSI 388? (They should.)

Minute Essay From Last Lecture

Slide 2

- Most people thought pace was pretty much okay, though having more of a chance to practice in class would be good. So — maybe.
- Several people aren't clear on the roles of various tools. Brief review shortly.
- "What language do I learn after C"? Depends on what kind of programs you want to write! ask me, or ask someone who's doing a kind of programming that appeals to you.
- One person has a developer friend who on hearing we were learning `vim` said "good luck with that". Yeah. People love it or they hate it.

Slide 3

Programming Basics and C

- Previous lecture describes relationship between what humans write (“source code”) and what computers execute (“machine language”).
- For C, usual process is that you write source code, and then it must be transformed not just into machine language, but into a complete “executable file” (machine language for your code, plus machine language for any library functions, plus information so operating system can load it into RAM and start it up). (Detail: This is for “hosted environment”; in some environments in which C is used, there may be no o/s.)
- So, what happens to your code . . .

Slide 4

Programming Basics and C, Continued

- Your code is first “compiled” into “object code” (machine language).

  Then it’s “linked” with any library object code to form “executable file”.

  Sometimes (as for examples we’re doing now) both steps happen as a result of a single command.
- To recap, we’re basically using two tools, `vim` (to write/edit programs) and `gcc` to compile them, and a command-line environment (terminal window) to run both tools and also programs we write.

Slide 5

“Hello World” Program Revisited

- Look again at the program we wrote in class previously. Most of it is standard boilerplate, to be discussed further soon. Single line you should pay attention to now is the one with `printf`.
- Goal for today — describe how to extend this to get input from “standard input” (keyboard by default), do simple computing, write results to “standard output” (terminal window by default).

Slide 6

Variables in C

- In C as in most/many other programming languages, you need temporary storage for data — e.g. someplace to save an input value and/or intermediate results. For this we use *variables*.
- Again in C as in many others — *variables*. In C variables must be *declared*, each with both a name and a *type*. Effect of declaring a variable is to reserve RAM for a value of the specified type and give it a name that can be referenced. (Similar to Matlab, except for choice of types?) What a name can look like is somewhat restricted (see textbook).
- Types in C are pretty basic — integers, “floating-point numbers” (for now, real numbers), and characters. Integer types are represented as fixed-size binary numbers and include various “sizes”. More about the others later.

Variables in C

Slide 7

- Variables are given values by *assignment statements* (using =, which here means “assign value on right to variable on left” rather than equality as in math!).
- Okay to change value with repeated assignments.

Expressions in C

Slide 8

- What’s on the right side of an assignment — *expression*.
- Expressions in C are similar to those in math, with some differences/extensions, partly due to limited range of symbols and partly due to how hardware usually works:

  `*` and `/` for multiplication and division; on integers division produces *quotient only*; to get remainder use `%`.
- An expression has a *value*, which is determined by *evaluating* it. Evaluation may have *side effects* — e.g., `printf("hello\n")` is an expression, with the side effect of “printing” and a value that often is not used.

Slide 9

Assignment Statements Revisited

- Simplest programs are often basically a sequence of assignment statements (plus some “statements” that are really just expressions, such as that `printf` in the “hello world”program).
- Unless otherwise indicated, statements are executed in the order in which they appear in the code. (Sequential-ness is important and sometimes trips up beginners.)

Slide 10

Simple I/O in C

- Use `printf` to display predefined text and values of variables. Syntax is that of “function call” (more later) with first parameter a “format string” that may include “conversion specifications”. Followed by zero or more expressions, one for each conversion specification. When statement is executed, expressions are evaluated and the results turned into something printable using those conversion specifications.
- Use `scanf` to get input. (It’s not really very good for interactive programs, but it’s what almost all intro texts use, so we will too, but keep in mind that it has limitations and annoyances.) Syntax very similar to that of `printf` except that rather than expressions you have *pointers* that say where to store value(s). More about pointers later; for now usually name of variable preceded by `&`.

Slide 11

Simple Examples

- Recap from last time: Compile (and link) with `gcc` I recommend *ALWAYS ALWAYS* compiling with optional flag `-Wall` so you get most optional warnings — sometimes annoying, but often very helpful! Example

  ```
  gcc -Wall hello.c)
  ```

  Then execute with `./a.out`.
- (Examples as time permits.)

Slide 12

Example — “Counting Change”

- Problem statement: Given a number of pennies, show how to represent it with minimum number of coins (pennies, nickels, etc.).
- (To be continued.)

Minute Essay

- Any questions? How similar is all of this to something you've used before, such as Matlab?