CSC161 2011S *Imperative Problem Solving*

Class 30: Pointers

Held: Tuesday, 15 March 2011

Summary: We begin our study of pointers, one of the more important data types in C. Pointers support the concept of memory as a first-class value.

Related Pages:

- EBoard.
- Lab: Pointers.
- Reading: K&R 5.1-5.2.

Notes:

- I have responded to most of the questions on the first reading on pointers.
- Tonight's reading: K&R 5.3-5.4.
- Break reading (or skimming): *Managing Projects with GNU Make*, Chapters 1 and 2.
- HWs 5 and 6 returned.

Overview:

- Memory as a First-Class Value.
- Examples from K&R.
- Detour: Writing Type-Generic Code.
- Lab.

Memory and Pointers

- "Everything" (or at least every variable) has to be stored somewhere in memory.
- The program needs to know where it's stored.
- In C, you can access information about where things are stored.
  - In Scheme (at least standard Scheme), you cannot.
  - We say that "memory locations are first-class values in C".
- The address-of operation (`&`) tells you where in memory that thing is stored.
- An address is also called a *pointer*.
- The type of a pointer is specified by the type of the thing pointed to, followed by a `*`. So `int *` means "a pointer to an integer".
- You can find the contents of something pointed to by prefixing the pointer with `*`.
- One of the most common uses of pointers is to permit functions to modify their (non-array) parameters. If you pass a pointer, you can dereference the pointer (get its contents) and modify those contents.

K&R Examples

- Many can be found in `Examples/KR5/`.
- `getint`
- `swap`

Detour: Writing Type-Generic Code

- There are lots of possible versions of `swap`.
- Do we really want to use copy-paste-change? **No!**
- What's the alternative? Use the C preprocessor.
- We write generic code in which
  - The type is replaced by TYPE
  - The name is replaced by PREFIX(num)
- For a particular type, we define those two macors.

Lab

- Do The Lab.
- Be prepared to reflect.
