CSE 374
Programming Concepts & Tools

Brandon Myers

Winter 2015

C: structs, linked lists, hw5

(Thanks to Hal Perkins)

Where we are

- We've seen most of the basic stuff about C, but we still need to look at structs (aka records or objects without methods) and linked data structures
  - Understand the code posted with today's lecture; we won't have time to walk through all the details
- Next: Rest of the C preprocessor (# stuff, macros), building multi-file programs
- Then: more programming tools (make, git)
- That will set us up for the next programming project
  - Which will start right after Monday's midterm

structs

- A struct is a record (i.e., a collection of data fields)
- A pointer to a struct is like a Java object with no methods
- `x.f` is for field access. (if is `x` not a pointer – new!)
- `(*x).f` in C is like `x.f` in Java. (if x is a pointer)
- `x->f` is an abbreviation for `(*x).f`
- There is a huge difference between a struct (value) parameter and a pointer to a struct
- There is a huge difference between local variables that are structs and those that are pointers to structs
- Again, left-expressions evaluate to locations (which can be whole struct locations or just a field's location)
- Again, right-expressions evaluate to values (which can be whole structs or just a field's contents)

C parameters - revisited

- C has a uniform rule for parameters (almost): When a function is called, each parameter is *initialized* with a *copy* of the corresponding argument (int, char, ptr,…) ("pass by value")
  - This holds even for structs! – a copy is created
  - There is no further connection between the argument and the parameter value in the function
    - But they can point to the same thing, of course
- ***But remember***: if the argument is an array name, the function parameter is initialized with a pointer to the array argument instead of a copy of the entire array
  - Implicit array promotion (we've seen this)

struct parameters

- A struct argument is copied (call-by-value)
- It is far more common to use a pointer to a struct as an argument instead of copying an entire struct
  - Gives same semantics as Java object references
  - Usually what you want – pointer to data that lives outside the function
    - Also avoids cost of copying a possibly large object
  - But occasionally you want call-by value (small things like complex numbers, geometric points, …)
- Puzzle: if an argument is an array containing a single struct, is it copied or is it promoted to a pointer?
  - What if it's a struct containing only a single array?

Linked lists, trees, and friends

- Very, very common data structures
- Building them in C
  - Use **`malloc`** to create nodes
  - Need to use casts for “generic” types
  - Memory management issues if shared nodes
  - Usually need to explicitly free entire thing when done
  - Shows tradeoffs between lists and arrays

- Look at the sample code and understand what it does/how it does it

HW5: T9

- (see hw5.html)