C: Functions and Pointers

CS 62 - Spring 2016
Michael Bannister

Pointers!

Address-of operator

- In C we will work directly with memory
- Place a `&` in front of a variable to get its location in memory
- Example: If `int x = 4;`, then `&x` is the location in memory where `4` is stored

Pointers!

Pointer Variables

- A variable that holds a memory address, similar to Java references
- Place a in front of a variable to declare the variable as a pointer variable
- Example: If int `*p`, then p is a variable for the memory location of an int
- Example: `p = &x` is a valid assignment

Pointers!

Dereferencing a pointer

- Place a `*` in front of a pointer to access the value at that memory location
- Example: `*p` is the int value `4`.

Example Code

- Example 5: Pointers 1
  - Some experiments with pointers

Void Pointers

Pointers of type `void*` can point to any memory location and provide no type safety.

Arrays

Arrays

- Contiguous block of memory
- Ignorant of their own size
- Variable holds address to first element

Assignment

- Assignment copies address NOT the array's elements
- Need to use a loop to do a "deep copy"

Arrays as function arguments

- Functions with array arguments must have an additional argument for the size of the array
- Functions can modify the entries in their array arguments just like with pointers

Example Code

- Summing the elements in an array
  - A lesson in error messages

Separate Compilation

Header files (`*.h`)

- Contain declarations and constant definitions
- "Copied" into files with the `#include` directive
  `#include < ... >` for system headers and
  `#include " ... "` for user headers
- Cannot be included twice; use guards
  (see example)

Separate Compilation

Implementation files (`*.c`)

- Contain the definitions of the the items declared in the corresponding header files, i.e., `my_functions.c` would contain the definitions of the items in `my_functions.h`.

A Tour of the Standard Library

Take a look at cppreference.com