C: Functions and Pointers

CS 62 - Spring 2016
Michael Bannister

Assignments

- Assignment: Continuing BooOS
  - A sample design will be posted today
  - Questions for the write up will be posted today
- Lab: Basic Input/Output in C
  - Lab write up will be post today.

Typedefs and structs

Current

```
struct point {
   int x;
   int y;
};
void point_print(struct point p) { ... }
```

Better

```
typedef struct point {
   int x;
   int y;
} point;
void point_print(point p) { ... }
```

Function Declaration vs. Definition

Function declaration

- Specify (1) function name; (2) return type; (3) types of all arguments
- Example: `int sum(int x, int y);`
- Functions must be declared before their first use

Function definition

- Specify the exact computation of the function
- Example: `int sum(int x, int y) {return x+y;}`
- Implicitly declare a function

Example Code

- Example 4: Sums 2
  - Forward declarations
  - Recursion

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

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

C Strings

- Array of characters: `char my_string[];`
- Strings are NULL terminated:
  "Hello" = {'H', 'e', 'l', 'l', 'o', '\0'}
- The array length is one more than the string length
- The type of a string is often written as: `char*`