Lecture 31: Functions and Pointers

Fall 2016

Kim Bruce & Peter Mawhorter

Midterm Postponed!

- Hopefully everyone saw the Piazza announcement
- If Monday is a problem for you, let us know ASAP and we'll arrange something
- If you requested some kind of accommodation and still want it, get in touch

Example Code

- `sum.c`
- `vector.c`
- `big.c`

Typedefs and Structs

```
// This is cumbersome
// must write 'struct point p'

struct point {
  int x;
  int y;
};
```

Typedefs and Structs

```c
struct point_s;
typedef struct point_s point;

struct point_s {
  int x;
  int y;
};
```

Declaration vs. Definition

- A *declaration* says “this will exist”
  - Specify type and name of variable
  - Specify name of struct
  - Specify return type, name, and argument type(s) of function
- It may exist in a different file (‘`external`’)
- A *definition* fills out details (value of a variable, what’s in a struct, etc.)
  - A *definition* implicitly *declares* what it *defines*.

Assignment

- Assignment always makes a copy
- Understand *exactly* what you are copying
- Assignment happens more often than you’d expect

Call-by-Value

- Functions are *call-by-value*: they get a copy of their arguments
- Modifications to a function’s arguments are invisible to the rest of the program.

Objects with Call-by-Value

- No methods → functions don't have special access to objects
- Call-by-value → functions can't access objects via arguments

So how can we modify an object?

Pointers!

- In C we work directly with memory
- '&' address-of operator returns a memory location
- If `int x = 4;` then `&x` is the location in memory where the 4 is stored

Pointer Variables

- `int *x` holds a "pointer-to-an-integer"
  - In Java, all of our Object variables were pointers
- '*' denotes a pointer variable
  - You can have a pointer to a pointer e.g., `int **x`
  - Pointer variables can hold references:
    `int x = 4; int *p = &x;`
  - The '*' applies to only one name:

```
int *x, y, *z;
```