CSE 374: Programming Concepts and Tools

Eric Mullen

Spring 2017

Lecture 14: Data Structures in C

Administrivia

- Midterm on Friday! During class time: bring something to write with that's *not* a red pen
    - Review session last night, last bit of today's lecture will be Q&A as well
- Late Days: keep track of them, once they're gone late work is worth nothing
- Thursday TA office hours have moved to 3:30-4:30pm

Today

- Types in C: what you have to work with, how to make more
  - Structs
  - Parameter passing
  - Typedefs
  - Casts
  - Signed/Unsigned: twos complement
- Q&A for Midterm

Structs

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

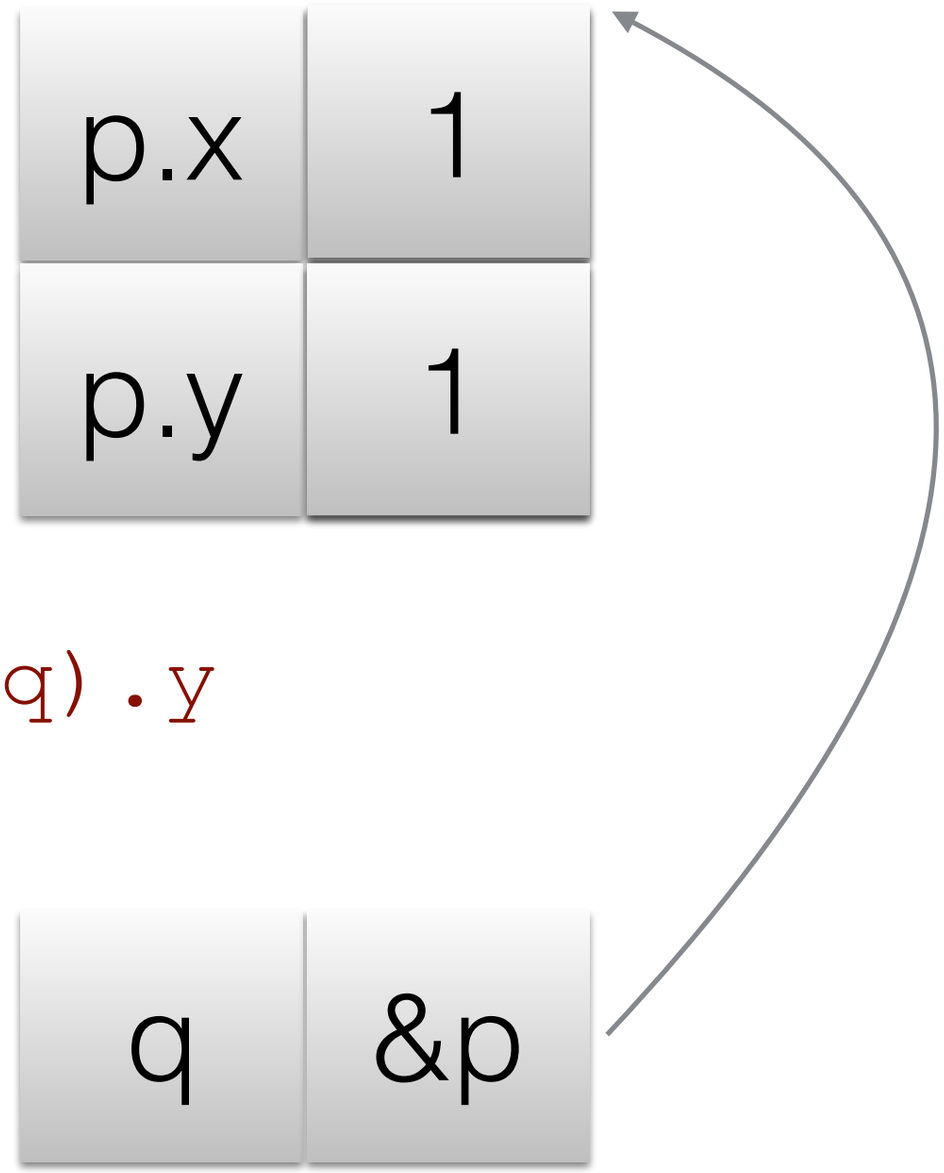
tag

fieldname

Annoying but necessary

Structs

```
struct point p;
p.x = 0;
p.y = p.x;
struct point* q = &p;
(*q).x = 1;
q->y = 1; //q->y means (*q).y
```



Parameters

- When parameters are passed, they're copied
  - Pointers are copied as well
  - Even structs are copied!
- Arrays are promoted to pointers, and the pointer value is copied

Struct Parameters

- Struct arguments are copied: can be expensive for large structs
- Much more common is to pass a pointer to a struct
- Likewise, you can return a struct, but common practice is a pointer to a struct (usually on the heap)

Data Structures

- You can make your favorite data structures in C!
  - Linked Lists
  - Trees
  - Stacks
  - Queues
  - …

Types in C

- char, int, float, double, long double, short, size_t
- void
- struct T (where T has already been defined)
- Array types (T[]) (easily promoted to pointers)
- Pointer types (T*)
- others (union T, enum T, function pointers)

typedef

- Just gives another name to a type

```
typedef int count;
```

- Can have weird consequences with Array types
- Works well with structs

typedef

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

struct point p;
point p;
```

Casts

- Sometimes you know more type information than C
  - e.g. with the result of `malloc`
- You can force C to give something a different type with a cast
  - To do so you just put the desired type in parentheses in front of an expression

Casts

```
int* x = (int*)malloc(sizeof(int)*32);

point p;
p.x = 0;
p.y = 0;
int* q = (int*)&p;
(*q) = 3; //p.x is 3
struct point *r = (struct point*)q;
```

Signed and Unsigned

- All numbers are stored as bits
- Some integer formats can have only positive values, some can have negative values
- `int` can have positive and negative
- `size_t` is only positive

Midterm Q&A

- Ask and you might get an answer :P