CS 241: Systems Programming
Lecture 16. Enums and Structs

Fall 2019
Prof. Stephen Checkoway

Announcements

Due to a (now corrected) misconfiguration, the deadline for homework 3 has been extended to 2019-10-18 (next Friday)

Fill out the form for group preferences today (right now!)

Course project

Work in groups of 4 (one group of 3)
- ‣ I'll assign the groups by Monday but I need your input **today**

A written proposal (1 page) is due in one week (Friday)

I'll give feedback on the proposal over fall break

A written status update (1 page) is due on November 15

The completed project and report (2 pages) is due on December 6

7 minute presentations will be the last week of class, December 9 and 11

Requirements

Must involve a significant amount of effort (more than one or two people could do alone)
- ‣ all partners are expected to contribute to the implementation, the write ups, and the presentation
- ‣ division of labor within each part is expected and good!

Involve a significant new programming technology you haven't used before
- ‣ a new language (C, C++, Rust, Go, Ruby, Haskell, JavaScript, etc.), or
- ‣ significant use of a framework or library (e.g., Django for Python or some graphical framework for Java)

Requirements

Collaboration must happen on GitHub
- ‣ This means regular commits
- ‣ I *strongly* recommend you learn about pull requests and use them along with code review of each commit before it gets pushed to master
- ‣ Using GitHub issues to track bugs that need to be fixed or features that need to be implemented is a great idea

Your code must contain tests similar to what we've done with the homework
- ‣ Find and use the appropriate tests for your language/framework
- ‣ You must use Travis CI to perform automated testing

What you do is up to you!

Suggestions
- ‣ Program a microcontroller (like an Arduino) to use some sensors and lights/actuators to do something (you can test with a simulator)
- ‣ Implement a game (e.g., Mancala, checkers) using some game building framework (SDL, PyGame, etc.)
- ‣ Use OpenCV to do something with computer vision
- ‣ Implement some machine learning algorithms (like k-means or k-nearest neighbors) and run them on some interesting datasets http://archive.ics.uci.edu/ml/index.php and do some visualization
- ‣ Build an interactive website
- ‣ Make some interactive art (audio and video)

Proposals

Due next Friday (2019-10-18) by the end of the day

Tell me
- ‣ what you are doing
- ‣ how are you doing it
- ‣ what you need to learn to do it
- ‣ a proposed schedule with milestones (the status report will discuss the milestones, you might want to track these on GitHub)

Be ambitious but **realistic**!
- ‣ Be explicit about which features are essential, which are nice-to-have that you plan to do, and which are stretch goals

Report

A two page (maximum!) write up
- ‣ standalone description of your project
- ‣ what you accomplished
- ‣ what you weren't able to get to
- ‣ what you found most challenging
- ‣ anything else you think I should know

Due the Friday before presentations (2019-12-06)

Demo and presentation

Last week of class (there will be a sign up for the day later in the semester)

Spend 7 minutes showing off and talking about your project
- 5 minutes of talking; 2 minutes of answering questions
- I know public speaking is *awful* (unless you enjoy it), but this is a super low-stakes way to get practice at it in a supportive environment
- Everybody must speak
- (Attendance at both days of presentations is mandatory, I will check with clickers)
- Tell us who you are, what you did, and how you did it (tell us what didn't work if you like)
- Show off some features
- 👏 👏 👏 Get some applause 👏 👏 👏

Enumerations: named constants

Anonymous, implicit values

- ‣ ```enum {```
  ```   FOO, // has value 0```
  ```   BAR, // has value 1```
  ```   QUX, // has value 2```
  ```};```
- ‣ These are integers
  ```int x = FOO;```

Named enums

You can name the enum

▸ ```
enum Color {
   RED,
   YELLOW,
   GREEN,
   /* etc. */
};
```
▸ This defines a new integer type
```
enum Color c = YELLOW;
```

Useful in switch statements
```
switch(c) {
case RED:
   return "red";
case YELLOW:
   return "yellow";
case GREEN:
   return "green";
/* etc. */
}
```
▸ Compiler can check you covered all cases

Explicit values

```c
enum Permission {
  READ_PERM  =  1 << 2,
  WRITE_PERM = 1 << 1,
  EXEC_PERM  =  1 << 0,
  RWX_PERM = READ_PERM | WRITE_PERM | EXEC_PERM,
};

/* We can use them as normal integers */
enum Permission no_exec(enum Permission perm) {
  return perm & ~EXEC_PERM;
}
```

```
enum Permission {
  READ_PERM =  1 << 2,
  WRITE_PERM = 1 << 1,
  EXEC_PERM =  1 << 0,
  RWX_PERM = READ_PERM | WRITE_PERM | EXEC_PERM,
};
```

What value does `EXEC_PERM + 4` have?

A. `READ_PERM`

B. `RWX_PERM`

C. `4`

D. `5`

E. Undefined behavior

Structures

Group related data together by creating a new type
```
struct Point {
 float x;
  float y;
};
```

Create and initialize a new Point named p
```
struct Point p = {
   .x = -33.8f,
   .y = 20.0f,
};
```
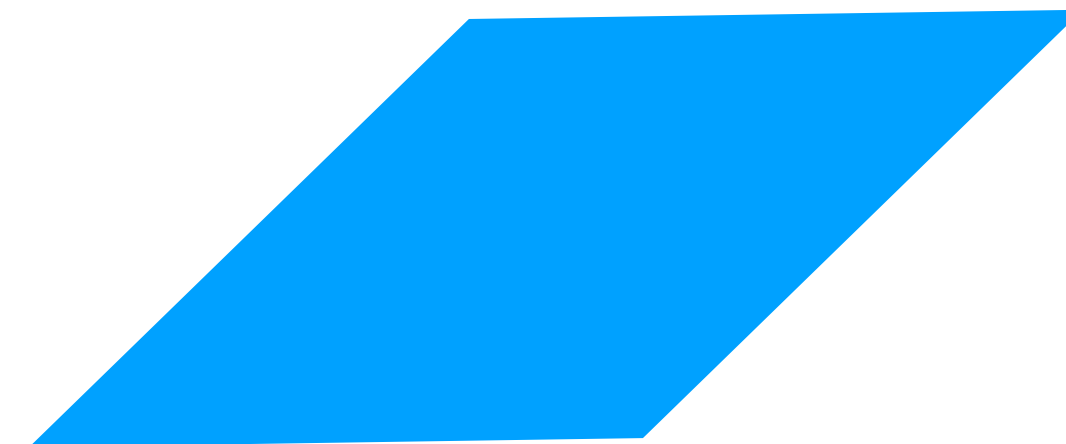
Nested structs

Structs can contain other structs (or arrays or arrays of structs or…)

```
struct Quadrilateral {
  struct Point vertex[4];
};
```

We can initialize a Quadrilateral

```
struct Quadrilateral rhombus = {
  .vertex = {
    [0] = { .x = 0.0f, .y = 0.0f },
    [1] = { .x = 1.0f, .y = 0.0f },
    [2] = { .x = 0.5f, .y = 1.0f },
    [3] = { .x = 1.5f, .y = 1.0f },
  },
};
```

Accessing a struct's members

struct `Point` has two members, `x` and `y`
- ‣ `p.x = 100.4f;`
- ‣ `printf("%f\n", p.y);`

struct `Quadrilateral` has one member `vertex` which is an array
- ‣ `rhombus.vertex` // gives a pointer to the first vertex
- ‣ `rhombus.vertex[3].x = 0.0f;`

C has structure values

We can pass a structure (by value) to a function or return one

```c
struct Quadrilateral embiggen(struct Quadrilateral q) {
    for (int i = 0; i < 4; ++i) {
        q.vertex[i].x *= 2.f;
        q.vertex[i].y *= 2.f;
    }
    return q;
}
```

```
struct Foo {
    int y;
};
struct Bar {
    int z;
    struct Foo f;
};
```

Given a **struct** Bar named b, how do you modify b's f's y member?

A. `struct Foo f = b.f;`
   `f.y = 35;`

B. `b.f.y = 35;`

C. `b.Foo.y = 35;`

D. Both A and B

E. All of A, B, and C

Pointers to structs

```c
// Use a pointer to a struct to update in place.
void embiggen2(struct Quadrilateral *q) {
  for (int i = 0; i < 4; ++i) {
    (*q).vertex[i].x *= 2.f; // Dereference q, then access vertex
    (*q).vertex[i].y *= 2.f; // Dereference q, then access vertex
  }
}


// Same as embiggen2, but using ->
void embiggen3(struct Quadrilateral *q) {
  for (int i = 0; i < 4; ++i) {
    q->vertex[i].x *= 2.f;
    q->vertex[i].y *= 2.f;
  }
}
```

```
struct Foo {
  int y;
};
struct Bar {
  int z;
  struct Foo *f;
};
```

Given a **struct** Bar named b, how do you modify b's f's y member?

A. `struct Foo *f = b.f;`
   `f->y = 35;`

B. `b.f->y = 35;`

C. `b->f.y = 35;`

D. Both A and B

E. All of A, B, and C

Anonymous structs

Like enums, structs can be anonymous
```
struct {
  char *const name;
  enum { STUDENT, GRADER, PROFESSOR } role;
} people[] = {
  [0] = { .role = PROFESSOR, .name = "Stephen" },
  [1] = { .role = GRADER,    .name = "Synthia" },
  /* ... */
};
```
‣ people is an array of this anonymous struct
‣ the role member is an anonymous enum
‣ note that the initializer need not list members in order
‣ we could make this whole thing const by writing **const struct**

Compound literals

Compound literal: (type) { initializer }
- ‣ (**struct** Point) { .x = 5.f, .y = 80.3f }

This has pretty limited use since you can just declare and initialize an instance of the struct

Macros are about the only time it's useful
```
#define MAKE_POINT(x_coord, y_coord) \
  (struct Point) { .x = (x_coord), .y = (y_coord) }
```

Type definitions

It's pretty clunky referring to things as enum Foo or struct Bar

Use a typedef!
- ‣ Generic form: **typedef** From To;
- ‣ Examples
  typedef struct Point Point;
  typedef enum Color Color;

You can typedef an anonymous struct
```
typedef struct {
    float x;
    float y;
} Point;
```

In-class exercise

https://checkoway.net/teaching/cs241/2019-fall/exercises/Lecture-16.html

Grab a laptop and a partner and try to get as much of that done as you can!