Software Systems

Day 11 – Structs, Testing

Structs

- In the reading you learned about structs, unions, and bitfields.
- Structs are by far the most commonly used, so we'll focus on those.

Structs: Typedef

- One rule that's helpful to stick to is to use typedef to declare and define structs:

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

- This saves you some verbosity, so you can write `pair` instead of `struct pair`.

Structs: Pointer Members and Const

- Structs can have members that are pointers:

```
typedef struct {
  char* name;
  unsigned int health;
} character;
```

- Be careful with const (`const char*` vs `char* const`):

```
char name[] = "Olin Man";
const character olin_man = {name, 42};
olin_man.name[1] = 'd';  // Still works
```

Structs: Padding

- You can get the number of bytes that a struct takes up with the sizeof operator:

```
typedef struct {
  int x;
  int y;
} pair;
printf("Size of pair: %zu\n", sizeof(pair));
```

- But this may not always behave the way you expect.

Structs: Padding

- You can use the `offsetof` function in `stddef.h` to find out how many bytes into a struct a member is:

```
typedef struct {
  char marker;
  int x_pos;
  int y_pos;
} point;
printf("x_pos is %zu bytes into point\n",
       offsetof(point, x_pos));
```

Structs: Padding

- Exercise:
  - Try defining structs with various members in it, then printing out its size.
  - (If you're stuck on where to start, try the character struct from earlier.)
  - Try to work out a rule for how the size of a struct is determined.
  - Where are the members located within the struct?
  - What happens when you have a struct inside a struct?

Testing

- We use Criterion in this class for testing.
  - (If you need mocking in your project, Mimick (by the same author) may be useful.)
- Your project should include some basic tests to make sure that things are working correctly.
- When in doubt, check old assignment files, or the official samples: https://github.com/Snaipe/Criterion/tree/bleeding/samples

Testing: Criterion Syntax

- Tests are defined with a suite name and a test name:

```
Test(suite_name, test_name) {
  // Write your test here
}
```

- The syntax above uses a preprocessor macro (like include guards).
  - So if you get macro-related warnings/errors, you know where to look.
- Suites are logically grouped collections of tests.
  - You can usually group tests by function and use its name as the suite name.

Testing: Criterion Syntax

- Within tests, you can assert things:

```
char* test_string = "Hello";
size_t test_size = 4;
cr_assert(eq(sz, strlen(test_string), test_size),
          "%s expected to be %zu characters",
          test_string, test_size);
```

- You can also assert Boolean values:

```
cr_assert(strlen(test_string) == test_size);
```

- But the output is less detailed.

Testing: Criterion Syntax

- There are many criteria you can use in tests:
  - `eq/ne`: equality/inequality
  - `lt/le/gt/ge`: less than (or equal to)/greater than (or equal to)
  - `zero`: 0 or equivalent value (e.g., NULL)
  - `ieee_ulp_eq/epsilon_eq`: compare two floats that are nonzero (ieee) or close to zero (epsilon)
- Each of these expects you to specify a data type (Tag):
  - The most common ones are sz (size_t), ptr (*), chr (char), int, and str (char*).
  - Lots of other numeric/string types in the docs.
  - Also a user-defined option if you want to write your own "methods".

Testing: Criterion Syntax

- To check stdout, you need to redirect it first:

```
Test(suite_name, test_name,
     .init = cr_redirect_stdout) {
  foo();  // Prints to stdout
  fflush(stdout);  // Make sure we wrote everything
  fclose(stdout);  // Close the file
  cr_assert_stdout_eq_str("bar");
}
```

Testing: Criterion Syntax

- You can also redirect stdin:

```
Test(suite_name, test_name) {
  FILE* stdin_redirect = cr_get_redirected_stdin();
  fprintf(stdin_redirect, "bar");
  fclose(stdin_redirect);
  foo();  // Consume from stdin
  cr_assert(...);  // Assert things here
}
```

Testing: Criterion Syntax

- Write a suite of tests to check the correctness of the following:
  - A function that takes an array of integers and returns the mean.
  - A function that takes an integer and prints it to stdout.
  - A function that reads up to a 3-digit integer from the user and returns it as an int.
- How you define the functions is up to you.
- Don't worry about writing CMake configurations for the tests.

Open Project Time

- Use this time to meet with your project team and/or work on your project.