STRUCTURES

ADAM SWEENEY
CS 211

INTRODUCTION

- We often want to represent something in our code that cannot be adequately represented by a single variable
- There's syntax for that

AGENDA

- What is even a structure
- Defining and working with structures
- Some uses and use cases

WHAT IS EVEN A STRUCTURE

WHAT IS IT GOOD FOR

- A way to collect many variables under a single name
- If this sounds like an array, it's because it does
  - But that's the extent
- Structures can hold data of different types!
  - This means that structs are heterogeneous data structures
  - Compare against arrays, homogeneous data structures

DEFINING AND USING STRUCTURES

THE EXAMPLE

- Let's say that we want to represent an album in our code
- Many pieces of data needed to represent an album
  - Artist (string)
  - Title (string)
  - Year (int)
  - Number of tracks (int)
  - Genre (string)
  - Rating (double)

DEFINE STRUCT

```
struct Album {
    std::string artist;
    std::string title;
    int year;
    int numTracks;
    std::string genre;
    double rating;
};
```

- What's familiar are the variable declarations
- The new syntax is that the declarations are placed inside a `struct Book` block
- This is a definition and NOT a declaration

DEFINE VS. DECLARE

- Define
  - Provide all information needed to create a "thing"
- Declare
  - Creates a thing and often sets properties

SOME USES AND USE CASES

INITIALIZING A STRUCT

- A `struct` can be initialized (assign value at declaration)

```
Album rb = {“Royal Blood”,
            “How Did We Get So Dark?”,
            2017, 10, “Rock”, 5};
```

- Note that the order of the data must be the same as they are listed in the `struct` definition

ASSIGNING TO A STRUCT

- A `struct` collects data under a single name
  - Like an array
- Unlike an array, we cannot use [] to access the members
- We must access `struct` members by name, using the dot operator

```
std::cout << book.title;
```

FUNCTIONS FOR STRUCTS

- We can have many functions to help us do things with our structures
  - Filling structs with data
  - Fill struct based on user input
  - Printing
  - Modifications
- Functions can return structs, and take them as parameters
  - Returning a `struct` is technically returning many values
    - Only a good idea when the data makes sense in a `struct`

A NOTE

- After writing all these functions to make working with structs easier, you might find code or be told that those functions can go *inside* the struct
- After all, these functions are likely going anywhere your struct goes
- They're not wrong
  - But they're wrong(-ish)
- In C++, the line between a struct and a class is very thin
  - We treat them differently out of principle and practice, not syntax

JUST BECAUSE…

- A struct can hold different data types
  - Doesn't mean it's the only way we should use them
- Sometimes we have data of the same type, but a `struct` is better suited
- One example is a name
  - If the first, middle, and last names are in their own strings it becomes easier to make certain changes
  - An array would be awkward since each element represents a different thing
  - The named members of a `struct` makes a lot more sense
  - Avoid "if all you have is a hammer"-itis