



Lecture 8

Structs I

What we will cover today

Structs in Rust

- Defining structs
- Instantiating structs
- Structs and ownership

Optional Reading:

The Rust Book Chapter 5.1 – Defining and Instantiating Structs

What are Structs?

- Short for "structure"
- Composite type – Holds multiple related values
- Similar concept to objects in OOP languages like Java
- Structs vs tuples
 - In a struct you can name each piece of data

Defining structs

```
struct Car {  
    brand: String,  
    make: String,  
    mpg: i32,  
}
```



fields

Keyword: **struct**

In each field:

- name followed by type

Instantiating structs

```
struct Car {  
    brand: String,  
    make: String,  
    mpg: i32,  
}
```

```
let my_car = Car {  
    brand: String::from("Toyota"),  
    make: String::from("Camry"),  
    mpg: 30,  
};
```

To instantiate a struct, we specify a concrete value for each field

Instantiating structs – Ordering of the fields

```
struct Car {  
    brand: String,  
    make: String,  
    mpg: i32,  
}
```

```
let your_car = Car {  
    mpg: 50,  
    make: String::from("Escape"),  
    brand: String::from("Ford"),  
};
```

The fields don't have to be specified in the same order as in the struct definition!

Dot notation

```
let mut my_car = Car {  
    brand: String::from("Toyota"),  
    make: String::from("Camry"),  
    mpg: 30,  
};
```

```
let my_car_mpg = my_car.mpg;  
my_car.brand = String::from("Chevrolet");  
my_car.make = String::from("Camaro");  
my_car.mpg = 100;
```

Use the dot notation to access the fields of the struct

Notice that now **my_car** has to be declared with **mut**!

Field init shorthand

```
let brand = String::from("Honda");  
let make = String::from("Civic");  
let mpg = 100;  
  
let your_car = Car {  
    brand,  
    make,  
    mpg  
};
```

When variables have the same name as the struct field, we can use it directly without using the **name: value** notation

This is useful in functions

Struct update syntax

```
let steal_car = Car{  
    mpg: 60,  
    ..my_car  
    // brand and make will be taken from my_car  
};
```


Initialize a struct using the value of another struct.

Specify the fields that we want. Remaining fields will be taken from the given struct.

Struct update syntax – Ownership changes!

```
let mut my_car = Car {  
    brand: String::from("Toyota"),  
    make: String::from("Camry"),  
    mpg: 30,  
};
```

```
let steal_car = Car {  
    mpg: 60,  
    ..my_car  
};
```

Two orange arrows originate from the right side of the first code block. One arrow points from the `brand` field to the `..my_car` update syntax in the second code block. The other arrow points from the `make` field to the same `..my_car` update syntax.

The value is moved! Now if we try to access the `brand` and `make` field of `my_car`, we'll get compiler errors.

Announcements

HW6 released today on PrairieLearn

Due 1 week from now — Next Friday 02/28 23:59