Structs

CS128 Honors

Slides by Matt Geimer (FA21)
Presented 9/29/2021

Why do we need structs?

```
fn main() {
    let student_name = "Ewan Knightley Stix";
    let student_uin = 128927128;
    let student_in_honors = true;
    let student_gpa = 4.0;

    printf!("{}",
       important_function(
         student_name,
         student_uin,
         student_in_honors,
         student_gpa
       )
    );
}
```

Why do we need structs?

- When we want to store related data, but they're not all the same types
- In the example given, a student has a
  - Name
  - UIN
  - Flag for if they're in honors
  - GPA
- These all are related to the same student, but separate variables

Declaring a struct

```
struct Student {
    name: String,
    uin: u32,
    in_honors: bool,
    gpa: f32
}
```

Instantiating a struct

```
struct Student {
    name: String,
    uin: u32,
    in_honors: bool,
    gpa: f32
}

fn main() {
   let student = Student {
       uin: 128927128,
       in_honors: true,
       name: String::from("Ewan Knightley Stix"),
       gpa: 4.0
    };
}
```

Accessing values in a struct

```
struct Student {
    name: String,
    uin: u32,
    in_honors: bool,
    gpa: f32
}

fn main() {
    let student = Student {
        uin: 128927128,
        in_honors: true,
        name: String::from("Ewan Knightley Stix"),
        gpa: 4.0
    };
    println!("Name: {}", student.name);
}
```

```
Name: Ewan Knightly Stix
```

Modifying values in a struct

```rust
struct Student {
    name: String,
    uin: u32,
    in_honors: bool,
    gpa: f32
}

fn main() {
    let mut student = Student {
        name: String::from("Ewan Knightley Stix"),
        uin: 128927128,
        in_honors: true,
        gpa: 4.0
    };
    student.gpa = 3.0;
    println!("GPA: {}", student.gpa);
}
```

Modifying values in a struct

- The **entire** struct must be **mutable or immutable**
- Individual values inside the struct cannot be individually mutable

Using a function to create a struct

```
fn new_student(name: String, uin: u32) -> Student {
    Student {
        name: name,
        uin: uin,
        in_honors: true,
        gpa: 4.00
    }
}
```

The reason our function only has two parameters is because two of the values are hard-coded to always be initialized to this

The type for the function is the struct name

- How does ownership factor into this function?

Student is instantiated, and its values are stored. `new_student` then gives ownership to the calling func

Using a function to create a struct

```
fn new_student(name: String, uin: u32) -> Student {
    Student {
        name: name,
        uin: uin,
        in_honors: true,
        gpa: 4.00
    }
}
```

These assignments are obvious, and Rust knows that too

Using a function to create a struct

```rust
fn new_student(name: String, uin: u32) -> Student {
    Student {
        name,
        uin,
        in_honors: true,
        gpa: 4.00
    }
}
```

Since these variables have the same name as the initializer variables, we can just pass them in

Struct update syntax

```
let student = Student {
    name: String::from("Ewan Knightley Stix"),
    uin: 128927128,
    in_honors: true,
    gpa: 4.0
};

let student2 = Student {
    name: String::from("Ewan Knightley Stix II"),
    uin: 858927128,
    in_honors: true,
    gpa: 4.0
};
```

- Say we have these two students
- How can we use some of the firsts' values in the second?

Struct update syntax

```
let student = Student {
    name: String::from("Ewan Knightley Stix"),
    uin: 128927128,
    in_honors: true,
    gpa: 4.0
};

let student2 = Student {
    name: String::from("Ewan Knightley Stix II"),
    uin: 858927128,
    in_honors: student.in_honors,
    gpa: student.gpa
};
```

- We could use dot notation, but that’s messy and annoying

Struct update syntax

```
let student = Student {
    name: String::from("Ewan Knightley Stix"),
    uin: 128927128,
    in_honors: true,
    gpa: 4.0
};

let student2 = Student {
    name: String::from("Ewan Knightley Stix II"),
    uin: 858927128,
    ..student
};
```

- A better way is struct update syntax
- Just use `..variable_name` to load the rest from that struct

Tuple Structs

- If you want to name a tuple, you can do so using a tuple struct
- Tuple structs are their own type, even if they hold the same data as another tuple struct
- They're declared like so:

```
struct Position(f32, f32, f32);
```

Structs

CS128 Honors

Slides by Matt Geimer (FA21)
Presented 9/29/2021