

**Equality & Ordering** | Comparing Structs

# ■ Comparison Operators

- ◆ Structs can be compared using equality operators
- ◆ Derivable traits enable structs to be compared
  - *PartialEq*
    - ▶ Provides equality
  - *PartialOrd*
    - ▶ Provides ordering: greater/less than
- ◆ *PartialOrd* requires *PartialEq* to be implemented

## ■ Example – *PartialEq*

```
#[derive(PartialEq)]
```

```
struct User {  
    id: i32,  
    name: String,  
}
```

```
let a = User { id: 1, name: "a".to_owned() };  
let b = User { id: 2, name: "b".to_owned() };  
  
if a == b {  
    //...
```

```
}
```

## ■ Example – *PartialOrd*

```
#[derive(PartialEq, PartialOrd)]
struct User {
    id: i32,
    name: String,
}

let a = User { id: 1, name: "a".to_owned() };
let b = User { id: 2, name: "b".to_owned() };
if a < b {
    //...
}
```

# ■ *PartialOrd*

- ◆ *PartialOrd* only considers the first struct field
- ◆ Manual implementation needed to compare other fields
  - Always ensure *PartialOrd* and *PartialEq* are consistent

# ■ PartialOrd – Manual Implementation

```
use std::cmp::Ordering;
```

```
impl PartialOrd for User {  
    fn partial_cmp(&self, other: &Self) -> Option<Ordering> {  
        if self.name < other.name {  
            Some(Ordering::Less)  
        } else if self.name > other.name {  
            Some(Ordering::Greater)  
        } else {  
            Some(Ordering::Equal)  
        }  
    }  
}  
}
```

```
#[derive(PartialEq)]  
struct User {  
    id: i32,  
    name: String,  
}
```

# ■ *PartialOrd* – Manual Implementation

```
use std::cmp::Ordering;
```

```
impl PartialOrd for User {  
    fn partial_cmp(&self, other: &Self) -> Option<Ordering> {  
        Some(self.name.cmp(&other.name))  
    }  
}
```

- ◆ *.cmp()* is made available through *#[derive(Ord)]*
  - Automatically derived on primitive types

```
#[derive(PartialEq)]  
struct User {  
    id: i32,  
    name: String,  
}
```

# ■ Recap

- ◆ Structs can be sorted and compared
  - *PartialOrd* and *PartialEq* implementation required
  - *Ord* implementation optional
- ◆ These traits can be used with *derive*
- ◆ Ordering respects only the first struct field when derived
  - Manual implementation required to order on different fields
  - Ordering and equality must remain consistent when implementing manually