

Stdlib | Operator Overloading

Operator Overloading

- ◆ Operators can be overloaded for structs and enums
- ◆ Trait implementation required
- ◆ All overloadable operators are available in *std::ops* module
- ◆ Behavior should be consistent with the meaning of the operator
 - Adding should make something larger
 - Subtracting should make it smaller, etc

■ Example – Add

```
use std::ops::Add;
```

```
struct Speed(u32);
```

```
impl Add<Self> for Speed {
```

```
    type Output = Self;
```

```
    fn add(self, rhs: Self) -> Self::Output {
```

```
        Speed(self.0 + rhs.0)
```

```
    }
```

```
}
```

```
let fast = Speed(5) + Speed(3);
```

■ Example – Add

```
use std::ops::Add;

struct Speed(u32);

impl Add<u32> for Speed {
    type Output = Self;

    fn add(self, rhs: u32) -> Self::Output {
        Speed(self.0 + rhs)
    }
}

let fast = Speed(5) + 3;
```

■ Example – Add w/Different Output

```
use std::ops::Add;

struct Letter(char);

impl Add<Self> for Letter {
    type Output = String;

    fn add(self, rhs: Self) -> Self::Output {
        format!("{}", self.0, rhs.0)
    }
}

println!("{}", Letter('h') + Letter('i'));
```

Common Operators

- ◆ *ops::Add* + lhs + rhs
- ◆ *ops::Sub* - lhs - rhs
- ◆ *ops::Mul* * lhs * rhs
- ◆ *ops::Div* / lhs / rhs
- ◆ *ops::Rem* % lhs % rhs
- ◆ *ops::Not* ! !item
- ◆ *ops::Neg* - -item

```
use std::ops::Index;
enum Temp {
    Current,
    Max,
    Min,
}
```

```
impl Index<Temp> for Hvac {
    type Output = i16;
```

```
    fn index(&self, temp: Temp) -> &Self::Output {
        match temp {
            Temp::Current => &self.current_temp,
            Temp::Max => &self.max_temp,
            Temp::Min => &self.min_temp,
        }
    }
}
```

```
struct Hvac {
    current_temp: i16,
    max_temp: i16,
    min_temp: i16,
}
```

```
enum Temp {  
    Current,  
    Max,  
    Min,  
}
```

```
let env = Hvac {  
    current_temp: 30,  
    max_temp: 60,  
    min_temp: 0,  
};
```

```
let current = env[Temp::Current];
```

Recap

- ◆ Operators are overloaded via traits
 - Listing of traits is in *std::ops* module
- ◆ Input type can be specified with generic parameter
- ◆ Output type can be specified with the *Output* associated type alias
- ◆ Behavior should remain consistent with operator purpose