Rust. Обобщённые типы

Выполнили: учащиеся группы: 5030102/00201

generics

- *generics* – абстрактные подставные типы, на место которых можно поставить любой конкретный тип
- Функции могут принимать параметры обобщённого типа для одинаковых действий над конкретными значениями

Удаление дублирования кода (1/2)

```rust
fn main() {
    let number_list = vec![34, 50, 25, 100, 65];

    let mut largest = &number_list[0];

    for number in &number_list {
        if number > largest {
            largest = number;
        }
    }

    println!("The largest number is {}", largest);

    let number_list = vec![102, 34, 6000, 89, 54, 2, 43, 8];

    let mut largest = &number_list[0];

    for number in &number_list {
        if number > largest {
            largest = number;
        }
    }

    println!("The largest number is {}", largest);
}
```

```rust
fn largest(list: &[i32]) -> &i32 {
    let mut largest = &list[0];
    for item in list {
        if item > largest {
            largest = item;
        }
    }
    largest
}

fn main() {
    let number_list = vec![34, 50, 25, 100, 65];
    let result = largest(&number_list);
    println!("The largest number is {}", result);
    assert_eq!(*result, 100);

    let number_list = vec![102, 34, 6000, 89, 54, 2, 43, 8];
    let result = largest(&number_list);
    println!("The largest number is {}", result);
    assert_eq!(*result, 6000);
}
```

Удаление дублирования кода (2/2)

```rust
fn largest_i32(list: &[i32]) -> &i32 {
    let mut largest = &list[0];
    for item in list {
        if item > largest {
            largest = item;
        }
    }
    largest
}
fn largest_char(list: &[char]) -> &char {
    let mut largest = &list[0];
    for item in list {
        if item > largest {
            largest = item;
        }
    }
    largest
}
fn main() {
    let number_list = vec![34, 50, 25, 100, 65];
    let result = largest_i32(&number_list);
    println!("The largest number is {}", result);
    let char_list = vec!['y', 'm', 'a', 'q'];
    let result = largest_char(&char_list);
    println!("The largest char is {}", result);
}
```

Console:

```
$ cargo run
   Compiling chapter10 v0.1.0
(file:///projects/chapter10)
error[E0369]: binary operation
`>` cannot be applied to type
`&T`
 --> src/main.rs:5:17
  |
5 |         if item > largest
{
  |            ---- ^ -------
&T
  |            |
  |            &T
  |
help: consider restricting
type parameter `T`
  |
1 | fn largest<T:
std::cmp::PartialOrd>(list:
&[T]) -> &T {
  |
++++++++++++++++++++++
```

```rust
fn largest<T>(list: &[T]) -> &T {
    let mut largest = &list[0];
    for item in list {
        if item > largest {
            largest = item;
        }
    }
    largest
}

fn main() {
    let number_list = vec![34, 50, 25, 100, 65];
    let result = largest(&number_list);
    println!("The largest number is {}", result);
    let char_list = vec!['y', 'm', 'a', 'q'];
    let result = largest(&char_list);
    println!("The largest char is {}", result);
}
```

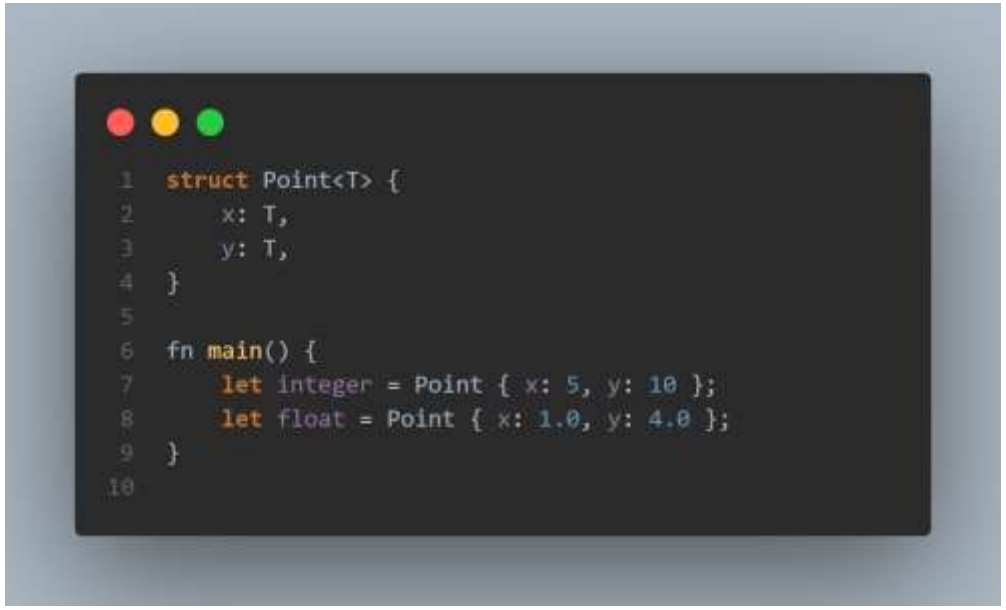
Ошибка:

функция не будет работать для всех возможных типов T

Типы, которые можно упорядочивать

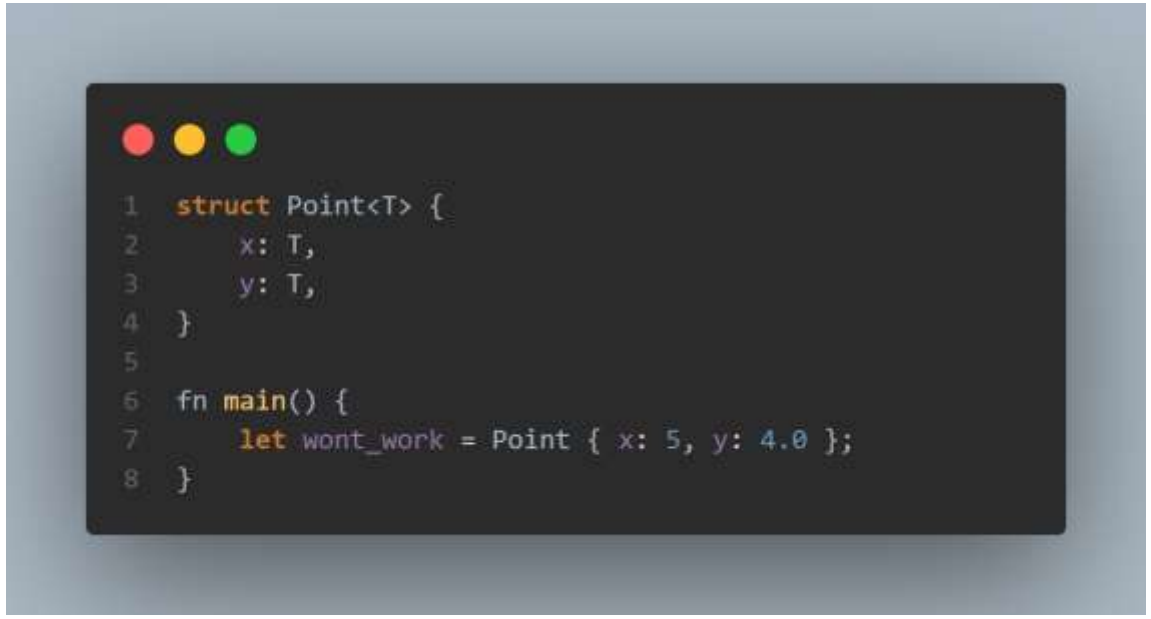
Для сравнений: *std::cmp::PartialOrd*

В определении структур (1/2)



```rust
struct Point<T> {
    x: T,
    y: T,
}

fn main() {
    let integer = Point { x: 5, y: 10 };
    let float = Point { x: 1.0, y: 4.0 };
}
```

- Использование похоже на синтаксис в определении функции
- Используем один тип => структура является обобщённой с типом T => поля x и y имеют одинаковый тип



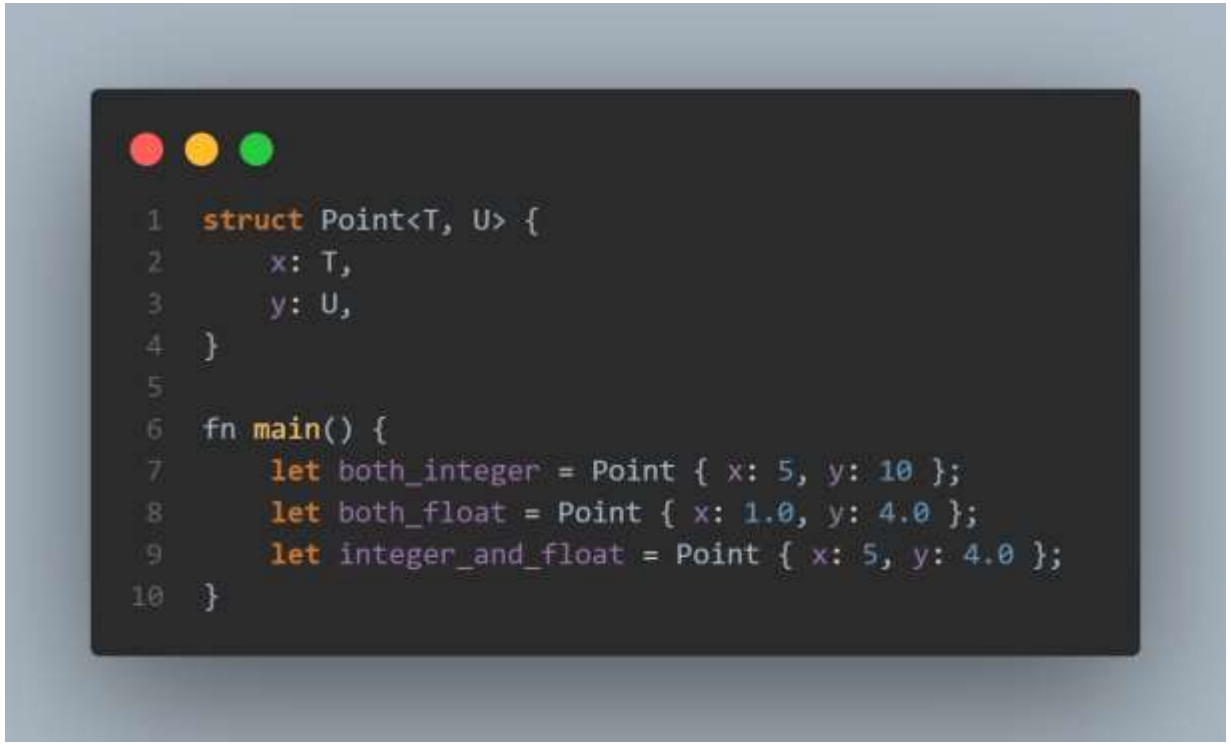
```rust
struct Point<T> {
    x: T,
    y: T,
}

fn main() {
    let wont_work = Point { x: 5, y: 4.0 };
}
```

Ошибка:

Переобозначение типов: сначала целочисленный, затем с плавающей точкой

```
$ cargo run
   Compiling chapter10 v0.1.0
(file:///projects/chapter10)
error[E0308]: mismatched types
 --> src/main.rs:7:38
  |
7 |     let wont_work = Point { x: 5, y: 4.0 };
  |                                      ^^^ expected
integer, found floating-point number
```

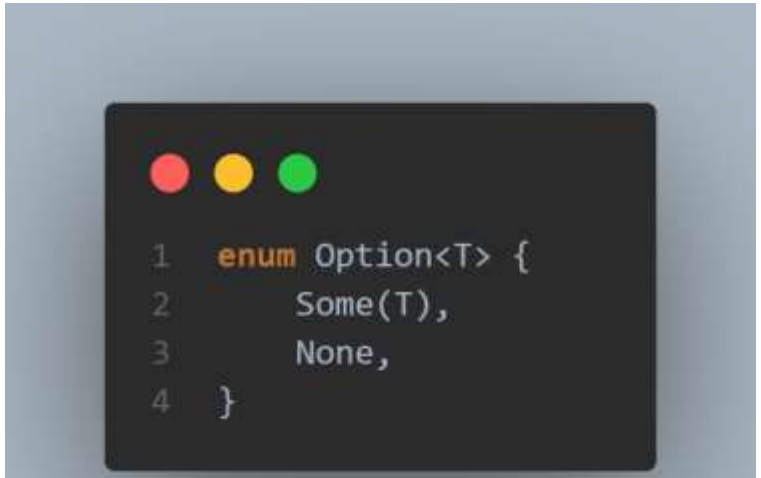
В определении структур (2/2)



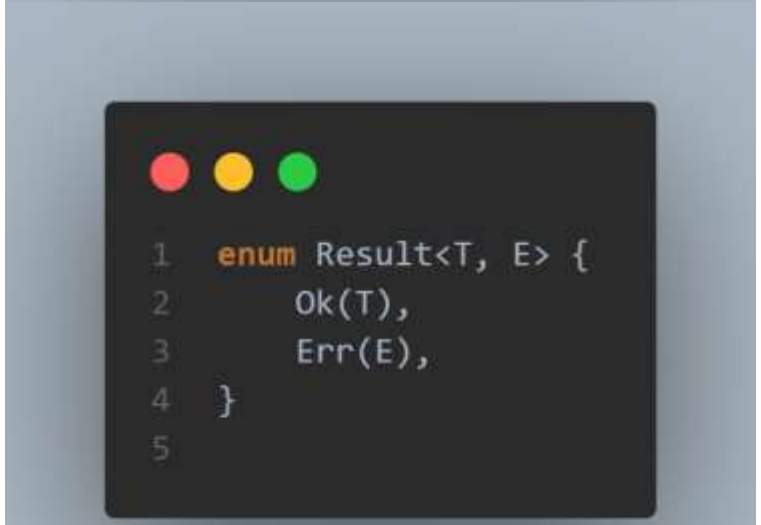
```rust
struct Point<T, U> {
    x: T,
    y: U,
}

fn main() {
    let both_integer = Point { x: 5, y: 10 };
    let both_float = Point { x: 1.0, y: 4.0 };
    let integer_and_float = Point { x: 5, y: 4.0 };
}
```

- Использование нескольких параметров обобщённого вида
  - x имеет тип T, y имеет тип U
- В объявлении можно использовать сколь угодно много параметров обобщённого типа
- Если их много => код трудночитаем => разбивка на более мелкие части

В определении перечислений



- Option<T> - перечисление из стандартной библиотеки
- Абстрактная концепцию необязательного значения
  - Содержит одно значение типа T / None (ничего не содержит)



- Операции выполнены успешно (неуспешно) => возвращают значение типа T (E)
- Пример: открытие файла
  - T - `std::fs::File` – файл открыт успешно
  - E - `std::io::Error` – возникли проблемы

В определении методов (1/2)

```rust
struct Point<T> {
    x: T,
    y: T,
}

impl<T> Point<T> {
    fn x(&self) -> &T {
        &self.x
    }
}

fn main() {
    let p = Point { x: 5, y: 10 };

    println!("p.x = {}", p.x());
}
```

Метод x, который возвращает ссылку на данные в поле x
Важно: объявляем T сразу после impl => Rust понимает, что тип в Point<..> является универсальным
Можем дать другое имя типа, отличное от имени в определении структуры

```rust
impl Point<f32> {
    fn distance_from_origin(&self) -> f32 {
        (self.x.powi(2) + self.y.powi(2)).sqrt()
    }
}
```

Ограничение: конкретный тип используется для определения метода (пр: только для Point<f32>)
Экземпляры типов отличные от f32 не будут иметь этот метод

В определении методов (2/2)

```rust
struct Point<X1, Y1> {
    x: X1,
    y: Y1,
}

impl<X1, Y1> Point<X1, Y1> {
    fn mixup<X2, Y2>(self, other: Point<X2, Y2>) -> Point<X1, Y2> {
        Point {
            x: self.x,
            y: other.y,
        }
    }
}

fn main() {
    let p1 = Point { x: 5, y: 10.4 };
    let p2 = Point { x: "Hello", y: 'c' };

    let p3 = p1.mixup(p2);

    println!("p3.x = {}, p3.y = {}", p3.x, p3.y);
}
```

Типы в определении структуры, не всегда совпадают с аналогами, использующимися в сигнатурах методов этой структуры.

Пример:

- Для Point используются типы X1, Y1
- Для метода mixup (создание нового экземпляра Point) – X2, Y2
- x – из self (тип X1); y – из Point (тип Y2)
- p3 – результат программы:
  - y типа char (т.к. Y взят у p2)

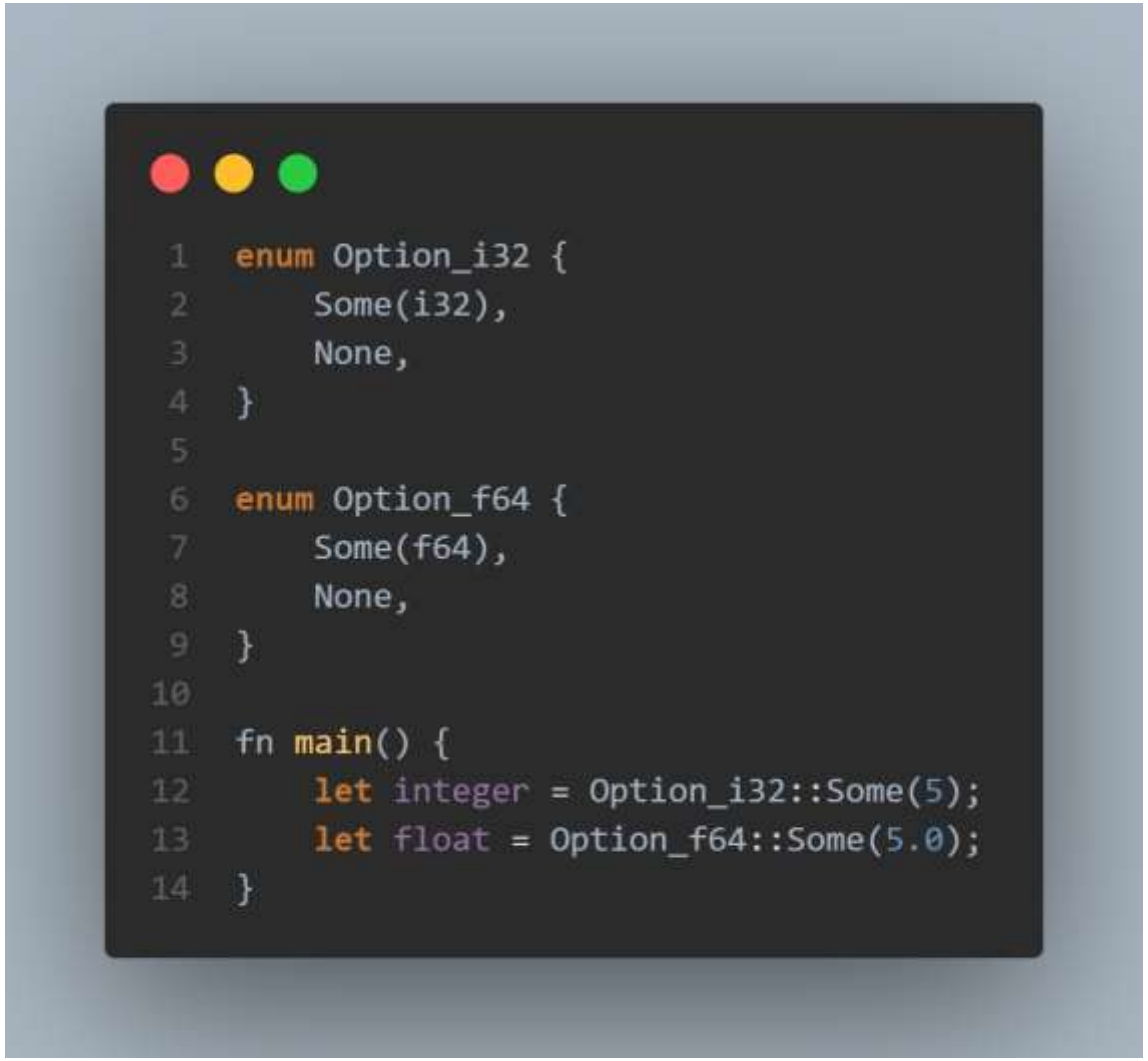
Console

```
p3.x = 5, p3.y = c
```

Производительность кода (1/2)

- Программа работает не медленнее, чем с использованием конкретных типов
- *Мономорфизация* – процесс превращения обобщённого кода в конкретный код путём подстановки конкретных типов, использующихся при компиляции.
  - Обратные шаги к созданию обобщённого кода
  - Смотрит места, где вызывается обобщённый код, и генерирует код для конкретных типов , использовавшихся для вызова в обобщённом.

Производительность кода (2/2)



```rust
enum Option_i32 {
    Some(i32),
    None,
}

enum Option_f64 {
    Some(f64),
    None,
}

fn main() {
    let integer = Option_i32::Some(5);
    let float = Option_f64::Some(5.0);
}
```

- При компиляции – компилятор считывает значения, которые были использованы в экземплярах Option<T> : для i32, для f64
- Заменяет конкретными определениями, созданными им
- Т.к. Rust компилирует обобщённый код в код, определяющий тип в каждом экземпляре, мы не платим за использование обобщённых типов во время выполнения.

Типажи как параметры (1/4)

- Вместо передачи конкретного типа можно передавать trait
- Для этого необходимо указать “impl” <trait_name> в качестве параметра
- Передаваемый тип должен реализовывать trait Summary

```
pub trait Summary {
    fn summarize(&self) -> String;
}
```

```
pub fn notify(item: &impl Summary) {
    println!("Breaking news! {}", item.summarize());
}
```

Типажи как параметры (2/4)

- Передавать типажи, как параметры можно и более подробным образом
- Такой способ является более полным, прошлый пример – синтаксический сахар
- Оба варианта работают одинаково

```
pub fn notify<T: Summary>(item: &T) {
    println!("Breaking news! {}", item.summarize());
}
```

```
pub fn notify(item1: &impl Summary, item2: &impl Summary) {
```

```
pub fn notify<T: Summary>(item1: &T, item2: &T) {
```

Типажи как параметры (3/4)

- Для использования нескольких типажей для одного аргумента существует синтаксис с оператором «+»
- Параметр “item” в данном примере должен реализовывать два типажа – «Summary» и «Display»

```
pub fn notify(item: &(impl Summary + Display)) {
```

```
pub fn notify<T: Summary + Display>(item: &T) {
```

Типажи как параметры (4/4)

- Помимо предыдущих примеров можно передавать аргументы более понятным образом

```
fn some_function<T, U>(t: &T, u: &U) -> i32
where
    T: Display + Clone,
    U: Clone + Debug,
{
```

Вместо

```
fn some_function<T: Display + Clone, U: Clone + Debug>(t: &T, u: &U) -> i32 {
```

Возврат значений, реализующих типаж

- Для возвращения типа, реализующего типаж используется следующий синтаксис с “impl”, реализуется полиморфизм
- Достаточно не указывать конкретный тип, а указать типаж

```
fn returns_summarizable() -> impl Summary {
    Tweet {
        username: String::from("horse_ebooks"),
        content: String::from(
            "of course, as you probably already know, people",
        ),
        reply: false,
        retweet: false,
    }
}
```

```
pub struct Tweet {
    pub username: String,
    pub content: String,
    pub reply: bool,
    pub retweet: bool,
}

impl Summary for Tweet {
    fn summarize(&self) -> String {
        format!("{}: {}", self.username, self.content)
    }
}
```

Типажи для создания методов

```rust
struct Pair<T> {
    x: T,
    y: T,
}

impl<T> Pair<T> {
    fn new(x: T, y: T) -> Self {
        Self { x, y }
    }
}
```

```rust
impl<T: Display + PartialOrd> Pair<T> {
    fn cmp_display(&self) {
        if self.x >= self.y {
            println!("The largest member is x = {}", self.x);
        } else {
            println!("The largest member is y = {}", self.y);
        }
    }
}
```

Материалы

- Официальная документация Rust(en, ru)
  - https://doc.rust-lang.ru/book/ch10-00-generics.html
- Программирование на языке Rust
  - https://vk.com/topic-51126445_36552642

Спасибо за внимание!