Własność obiektów i struktury

10 października 2025

Instrukcja

1. Zaimplementuj program opisany w kolejnej sekcji.
2. Napraw wszystkie ostrzeżenia kompilatora.
3. Znajdź powszechne problemy z kodem używając `cargo clippy` i je napraw.

Program

1. Zdefiniuj strukturę `NumberWithUnit`, która ma dwa pola: `unit: String` i `value: f64`.
2. Zaimplementuj dla `NumberWithUnit` dwie funkcje skojarzone pełnące role konstruktorów:
   (a) `unitless(value: f64) -> Self`,
   (b) `with_unit(value: f64, unit: String) -> Self`,
   (c) `with_unit_from(other: Self, value: f64) -> Self` (kopiuje jednostkę z `other`).
3. Przy pomocy atrybutu `derive` zaimplementuj dla `NumberWithUnit` cechy `Debug`, `Clone` i `Default`. Czy można też Copy? Co jeśli usuniesz ze struktury pole typu String? Dlaczego tak jest?
4. W funkcji `main` stwórz wartości wszystkimi napisanymi funkcjami skojarzonymi. Wyświetl je używając `println!`.
5. Dodaj do `NumberWithUnit` trzy metody:
   (a) `add(self, other: Self) -> Self`: dodaje wartości, jeśli jednostka jest taka sama, w przeciwnym wypadku panikuje (makro `panic!`).
   (b) `mul(self, other: Self) -> Self`: mnoży wartości i jednostki (np. jeśli pomnożono `7m` i `2s` to wynikiem jest `14m*s`).
   (c) `div(self, other: Self) -> Self`: jak wyżej, ale z dzieleniem.
6. Dodaj do `NumberWithUnit` trzy analogiczne metody, ale z sygnaturami
   (a) `add_in_place(&mut self, other: &Self)`,
   (b) `mul_in_place(&mut self, other: &Self)`,
   (c) `div_in_place(&mut self, other: &Self)`.
7. Użyj powyższych funkcji (punkt 5 i 6) w funkcji `main`, np. policz prędkość z odległości i czasu.
8. Napisz funkcję `mul_vals`, która przyjmuje slice `NumberWithUnit` i zwraca `NumberWithUnit` z wymnożonymi wszystkimi elementami slice.
9. Napisz funckję `mul_vals_vec` analogiczną do powyższej, ale przyjmującą `Vec` (nie `&Vec`!).

10. Użyj napisanych funkcji w funkcji `main`. Spróbuj wywołać na jednym wektorze dwukrotnie `mul_vals` i dwukrotnie `mul_vals_vec`. Dlaczego pierwsze dwa wywołania się kompilują, ale drugie wywołanie `mul_vals_vec` już nie? Popraw kod tak, by drugie wywołanie też się kompilowało bez zmieniania nagłówków funkcji—użyj do tego klonowania.

11. Zdefiniuj strukturę krotkową `DoubleString`.

12. Zdefiniuj dla `DoubleString` funkcje skojarzone pełniące rolę konstruktorów:

    (a) `from_strs(str_1: &str, str_2: &str) -> Self`,

    (b) `from_strings(str_1: &String, str_2: &String) -> Self` (clippy będzie mówił, że `&String` nie jest najlepszym typem dla argumentu—to możemy zignorować).

13. Zdefiniuj dla `DoubleString` metodę `show(&self)`, która wyświetla oba napisy w postaci `(napis_1, napis_2)`.

14. W funkcji `main` utwórz obiekty `string: String` i `str_slice: &str`.

15. Utwórz dwa obiekty typu `DoubleString`:

    (a) używając `from_strs`, `string` i `str_slice`,

    (b) używając `from_strings`, `string` i `str_slice`.

    i wyświetl je używając `show`. Dlaczego wywołanie `from_strs(&string, str_slice)` się kompiluje, a `from_strings(&string, str_slice)` nie? Dlaczego w pierwszym przypadku `string` wymaga `&` a `str_slice` nie?

Wymagania

- Kod kompiluje się przy użyciu stabilnego kompilatora Rust.
- Brak ostrzeżeń z kompilatora i ‘clippy‘.
- Pełna implementacja zadanej funkcjonalności

Ocena (3 pkt)

- 1 pkt: Praca w trakcie laboratorium.
- 1 pkt: Pełna funkcjonalność (pełna implementacja zgodna z wymaganiami).
- 1 pkt: Prezentacja rozwiązania i odpowiedź na pytania prowadzącego.