Ownership and structs

9 October 2026

Instructions

1. Implement the program described in the next section.
2. Fix all compiler warnings.
3. Find common problems in the code using `cargo clippy` and fix them (problems with the names `add`, `mul` and `div` may remain—these can be ignored).

Program

1. Define a struct `NumberWithUnit` with two fields: `unit: String` and `value: f64`.
2. Implement for `NumberWithUnit` associated functions serving as constructors:
   (a) `unitless(value: f64) -> Self`,
   (b) `with_unit(value: f64, unit: String) -> Self`,
   (c) `with_unit_from(other: Self, value: f64) -> Self` (copies the unit from `other`).
3. Using the `derive` attribute, implement the traits `Debug`, `Clone` and `Default` for `NumberWithUnit`. Can you also derive Copy? What if you remove the String field from the struct? Why is that?
4. In `main`, create values using all of the associated functions you wrote. Print them using `println!`.
5. Add three methods to `NumberWithUnit`:
   (a) `add(self, other: Self) -> Self`: adds the values if the units are the same, otherwise panics (the `panic!` macro).
   (b) `mul(self, other: Self) -> Self`: multiplies the values and the units (e.g. multiplying `7m` by `2s` gives `14m*s`).
   (c) `div(self, other: Self) -> Self`: as above, but with division.
6. Add three analogous methods to `NumberWithUnit`, but with the signatures
   (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. Use the functions above (items 5 and 6) in `main`, e.g. compute a speed from a distance and a time.
8. Write a function `mul_vals` that takes a slice of `NumberWithUnit` and returns a `NumberWithUnit` equal to the product of all elements of the slice.

9. Write a function `mul_vals_vec` analogous to the one above, but taking a `Vec` (not a `&Vec`!).

10. Use the functions you wrote in `main`. Try calling `mul_vals` twice and `mul_vals_vec` twice on the same vector. Why do the first two calls compile, while the second call to `mul_vals_vec` does not? Fix the code so that the second call compiles as well, without changing the function signatures—use cloning for this.

11. Define a tuple struct `DoubleString`.

12. Define associated functions serving as constructors for `DoubleString`:

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

    (b) `from_strings(str_1: &String, str_2: &String) -> Self` (clippy will say that `&String` is not the best type for an argument—this can be ignored).

13. Define a method `show(&self)` for `DoubleString` that prints both strings in the form `(string_1, string_2)`.

14. In `main`, create the objects `string: String` and `str_slice: &str`.

15. Create two objects of type `DoubleString`:

    (a) using `from_strs`, `string` and `str_slice`,

    (b) using `from_strings`, `string` and `str_slice`,

    and print them using `show`. Why does the call `from_strs(&string, str_slice)` compile, while `from_strings(&string, str_slice)` does not? Why does `string` need a `&` in the first case, while `str_slice` does not?

Requirements

- The code compiles with the stable Rust compiler.
- No warnings from the compiler or `clippy`.
- Full implementation of the specified functionality.

Grading (3 pts)

- 1 pt: Work during the laboratory.
- 1 pt: Full functionality (complete implementation meeting the requirements).
- 1 pt: Presentation of the solution and answers to the instructor's questions.