Day 04

Structifying the Naughty List

The Story

Santa burst into the dev lounge, chugging his third espresso. "Great job yesterday, nerds! The `is_nice` function? Chef's kiss. But now, I want structure. STRUCTURE! We're going full-on Rustacean. I need a `Kid` struct—immediately!"

The Story

The elves nodded enthusiastically, their tiny laptops open, running Arch Linux with bspwm because, obviously, they were that kind of devs. One elf, started yapping, "But Santa, why a struct? Isn't this just overengineered?"

The Story

Santa slammed the table, shaking an untouched tray of gluten-free cookies. "No! A struct means no more random strings floating around. We need to encapsulate a kid's data—name, and niceness score. Plus, we'll need some methods to make sense of it all."

The Story

The dev elves scrambled to work. In no time, they sketched out the basic blueprint. Santa glanced at the screen. "Not bad. But I will need this extended later. Keep it modular, bros!"

The room fell silent as the elves realized the implications. This was just the beginning of Santa’s unhinged data modeling spree.

Your Task

The elves need your help to finish the `Kid` struct.

Here is what you need to do:

- Add two variants to the `Niceness` enum: `Nice` and `Naughty`. `Nice` takes the number of good deeds.

Your Task

- Add two fields to the `Kid` struct: `name` of type `String` and `niceness` of type `Niceness`.
- Move the `is_nice` function we created on Day 3 to an associated function of the `Kid` struct.
- Finally, implement the `new()` associated function for the `Kid` struct.

Hints

If you're stuck, here are some hints to help you get back on track:

- Use `Nice(u32)` to represent the number of good deeds.
- To define a field in a struct, use the syntax `field_name: field_type`, e.g., `name: String`.

Hints

- An associated function is defined in the `impl` block without the `self` parameter.
- Associated functions are called with `::` instead of `.`, e.g. `Kid::is_nice(10, 1);`
- Use `Self::is_nice` or `Kid::is_nice` to call the associated function from within the `impl` block.