Day 23

The Naughty-Nice Glitch

The Story

"Santa, we need to talk!" Snowball stammered, holding a stack of printouts that smelled vaguely of desperation.

Santa looked up, his patience thinner than Rudolph's battery life. "What now?"

Bernard, the senior elf, didn't mince words. "It's the Naughty-Nice List system. Legacy code. Written in JavaScript."

The Story

Santa froze. "Not even TypeScript?"

Bernard shook his head. "Plain JavaScript. `var` everywhere. No types. No safety. It's a dumpster fire. Someone even polyfilled `Promise` with copy-paste."

Santa slammed his mug down. "So Naughty kids get PS5s, and Nice kids get socks because JavaScript?"

The Story

Snowball nodded, sweating. "It's worse, Santa. There's no validation. The Nice List has a `SELECT *` injection, and someone added `console.log("Merry Christmas, LOL");` in production!"

Santa pinched the bridge of his nose. "Rewriting it in Rust is our only hope. No globals, no runtime panics—safety guaranteed. Bernard, make it happen."

The Story

"But Santa," Bernard hesitated, "starting from scratch this late… it's risky."

Santa leaned forward, eyes blazing. "Riskier than trusting JavaScript on Christmas Eve? I don't care how late it is. Write it in Rust."

Your Mission

Implement a `SantaList` struct that uses a `HashMap` to store children's names as keys and their behaviors (`true` for nice, `false` for naughty) as values.

Your Mission

Fields

The `SantaList` struct should have a single field:

- `records` - a `HashMap<String, bool>` to store children's names and behaviors.

Your Mission

Methods

The struct should have the following methods and associated functions:

- `new` - Create a new `SantaList` instance.
- `add` - Add a child's name and behavior to the list.

Your Mission

Methods

- `remove` - Remove a child from the list.
- `get` - Retrieve a child's behavior.

Your Mission

Methods

- `count` - Count the number of nice and naughty children as a tuple `(nice, naughty)`.
- `list_by_behavior` - Retrieve a list of children based on their behavior as `Vec<String>`.

Hints

If you're unsure where to start, take a look at these tips:

- Using `HashMap`: Import it with `use std::collections::HashMap;`. Use `HashMap::new()` to create an empty map.
- Adding/Updating: Use `insert(key, value)` to add or update an entry.

Hints

- Querying: Use `get(key)` to retrieve values. It returns an `Option<bool>`.
- Removing: Use `remove(key)` to delete an entry and get its value if it exists.
- Counting: Use `values()` with `.filter()` to count nice/naughty children.