Day 07

The Great Log Hunt

The Story

The tension in the North Pole Dev room had not dissipated. Santa's absence loomed large, and Blitzen had clearly let the power go to his antlers. The elves were beginning to mutter about mutiny—especially after Blitzen had loudly declared that `grep` was overrated and "real devs" write their own search tools.

The Story

"We need a better log system," Blitzen announced, pacing in front of the DevOps board like a caffeinated startup founder. "I'm tired of manually combing through logs! It's time we automate this."

Prancer peeked up from their desk. "Can't we just pipe the logs into `grep` like everyone else?"

The Story

Blitzen's glare could have melted the polar ice caps. "Prancer, if you're going to suggest mediocre solutions, you can go back to working in Node.js."

Prancer recoiled, whispering, "Too far, Blitzen. Too far."

A LogQuery Tool

Blitzen wanted a log search tool so advanced that even Santa would call it "blitzening fast". Logs were piling up from every North Pole subsystem—Toy Tracker 3000, SleighOS, and even Reindeer AI. The elves needed to find specific entries without scrolling for hours.

A LogQuery Tool

"You!" Blitzen pointed at Frostbyte, the elf known for typing faster than a Model M keyboard. "You're going to write a `LogQuery` struct in Rust that can search through our logs."

A LogQuery Tool

Frostbyte cracked his knuckles, opened NeoVim, and got to work.

But he needs your help to be saved from Blitzen's sass and implement the `LogQuery` struct with its `search` method?

The requirements

Here's what Frostbyte must implement:

- **`LogQuery` struct:** This struct should hold a reference to a slice of `String` logs.
- **`new()`:** Create an associated function named `new` that accepts a reference to a `Vec<String>` and returns a `LogQuery`.

The requirements

- **Search method:** A method named `search` that:

1. Returns a `Vec` of references to strings containing the `keyword`.
2. Should work even if the logs are empty.
3. Special characters in the `keyword` must be handled properly.

Hints

If you're stuck or need a starting point, here are some hints to help you along the way!

- Your `LogQuery` struct will likely hold a `&'a Vec<String>`:

```
pub struct LogQuery<'a> {
    logs: &'a Vec<String>,
}
```

Hints

- The `new` function should accept a reference to a `&'a Vec<String>` and return a `LogQuery`.
- The `search` method should accept a `&self` and a `keyword: &str` parameter.
- To return references to the logs, you can use `-> Vec<&'a String>`
- Implement `search` by iterating over `self.logs`. e.g. `self.logs.iter()`

Hints

- Use `filter` and provide a closure. e.g. `filter(|log| {})`
- Use `contains` to check if a log contains the `keyword`. e.g. `log.contains(keyword)`
- Use `collect` to convert the iterator back to a `Vec`. e.g. `collect::<Vec<_>>()`