Day 13

Santa's error handling rant

The Story

Santa found it a good opportunity to have a look to see what Blitzen has been doing all this time when he was tech lead.

Rolling up his sleeves, he ran a `git diff` command and began scrolling through the commits. His candy-cane-themed monitor reflected the code as his eyes narrowed like a debugger catching a null pointer. Then he froze, staring at the `parse_row` function.

The Story

"Blitzen," he barked, "what is this disaster? return `&'static str` for errors? Seriously?!"

Blitzen shrugged, sipping his oat milk latte. "MVP, Santa. Move fast, y'know?"

The Story

Santa slammed his mug of hot coffee down, marshmallows flying. "MVP? Last time you were in charge, you tried to rewrite grep and cat in Rust because you were 'bored'! And now you're telling me you can't handle proper error types? It takes five minutes! Do you want runtime panics for Christmas?!"

The Story

Bernard leaned in. "What if we use an `enum`? Variants for different errors, match on them later. Clean, ergonomic."

Santa's face lit up like Rudolph's nose. "Finally, a solution that doesn't make me want to refactor the North Pole. Bernard, you get an extra cookie ration this year. Blitzen? No oat lattes for a week."

Blitzen sighed. "Enum it is."

Your Mission

- Create an `enum` called `ParseError` with a few variants:
  - `NoName`
  - `NoGoodDeeds`
  - `NoBadDeeds`
  - `InvalidGoodDeeds`
  - `InvalidBadDeeds`

Post-mission

- When errors displayed, they should be human-readable:
  - `ParseError::NoName` should display as `"Name field is missing"`
  - `ParseError::NoGoodDeeds` should display as `"Good deeds field is missing"`
  - `ParseError::NoBadDeeds` should display as `"Bad deeds field is missing"`
  - `ParseError::InvalidGoodDeeds` should display as `"Good deeds value is invalid"`
  - `ParseError::InvalidBadDeeds` should

Your Mission

- Implement the `std::error::Error` trait for `ParseError`.
- Update the `parse_row` function to return meaningful errors using the `ParseError` enum.

Your Mission

- Make sure you return the correct error variant for each error condition.
- Handle the cases where an item in a row is missing. e.g. `Alice,,3` in this case, you should return an error with the `NoGoodDeeds` variant.

Hints

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

- The `Error` trait requires you to implement the `Display` and `Debug` traits for the custom error type.
- Implement `Debug` by using the `derive` macro. e.g. `#[derive(Debug)]`

Hints

- For `Display` you must implement manually. e.g.

```
use std::fmt;

impl fmt::Display for ParseError {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        match self {
            ParseError::NoName => write!(f, "No name found"),
            // other variants...
        }
    }
}
```

Hints

- Now that both traits are implemented, you can implement the `Error` trait.

```
use std::error::Error;

impl Error for ParseError {}
```