Creating Parsers with Rust nom Crate

Kat

What is this talk about?

Raising awareness of the nom crate

Will provide overview and code samples

Encourage you to read the full documentation

What is nom?

A parsing toolkit for Rust.

It is a parser library.

- A parser combinator library.
- A system for creating complex parsers by combining simple parsers.

Great replacement for using regular expressions :P

- Nom parsers are Rust functions
- Nom is NOT based on grammars or generators

`Parser` **Trait**

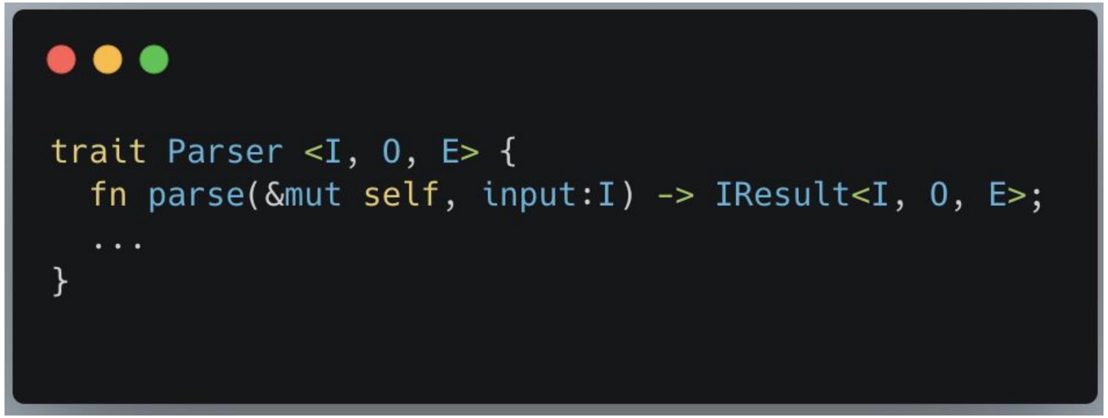
All nom parsers fulfill the parser trait

    This describes the execution of a single parse

Parser trait is automatically implemented for any function that returns an `IResult`

3 generic types

- Input
- Output
- Error

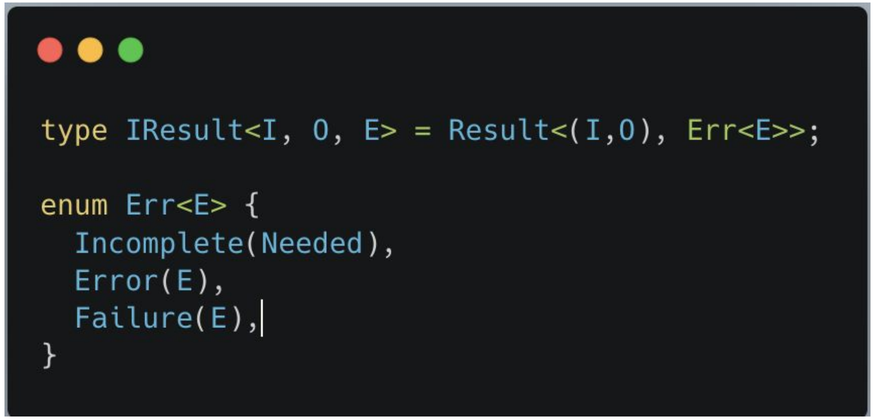


```
trait Parser <I, O, E> {
  fn parse(&mut self, input:I) -> IResult<I, O, E>;
  ...
}
```

`IResult` **Type**

Generic type that encodes the result of the parser

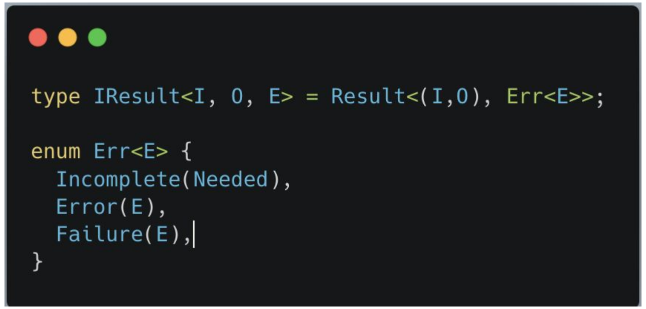
Allows for flexible error handling in parsing



```
type IResult<I, O, E> = Result<(I,O), Err<E>>;

enum Err<E> {
  Incomplete(Needed),
  Error(E),
  Failure(E),
}
```

Possible Parse Outcomes



```
type IResult<I, O, E> = Result<(I,O), Err<E>>;

enum Err<E> {
  Incomplete(Needed),
  Error(E),
  Failure(E),
}
```

Success: `Ok((I, O))`

- Parse succeeded
- Returns output (specific to parser) and returns unparsed tail of the input

Error: `Err(nom::Err::Error(E))`

- Parser encountered an error
- How the errors are handled is key to nom’s ability to allow components to be combined

Failure: `Err(nom::Err::Failure(E))`

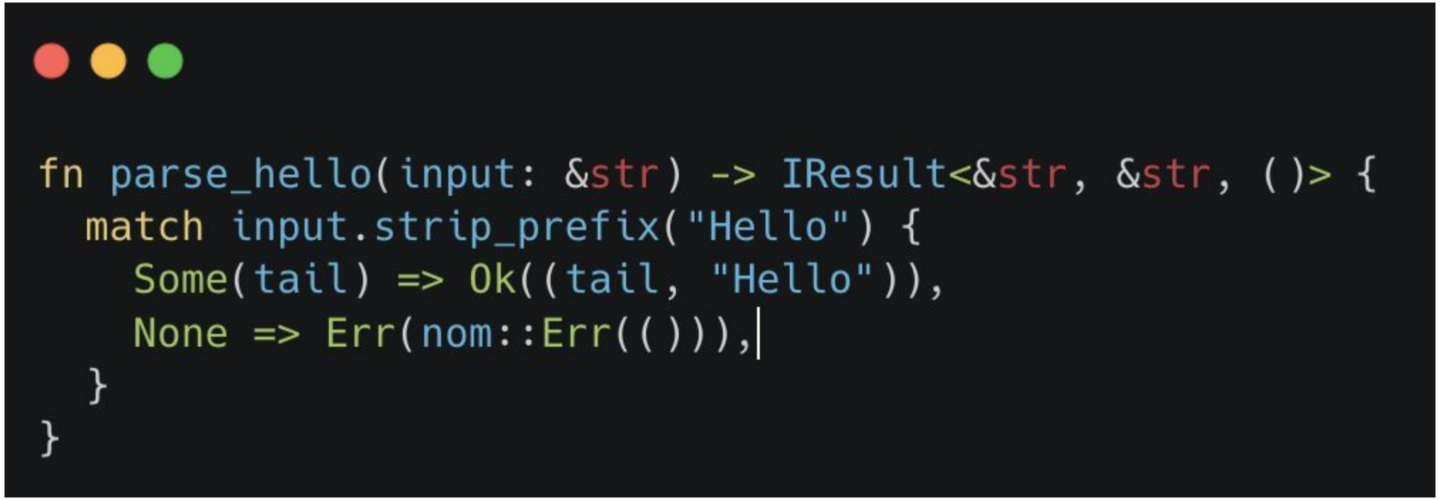
- Parser encountered an unrecoverable error

Incomplete: `Err(nom::Err::Incomplete(Needed))`

- Parser did not have enough information to succeed
- Only relevant to streaming parsers

Basic Parser

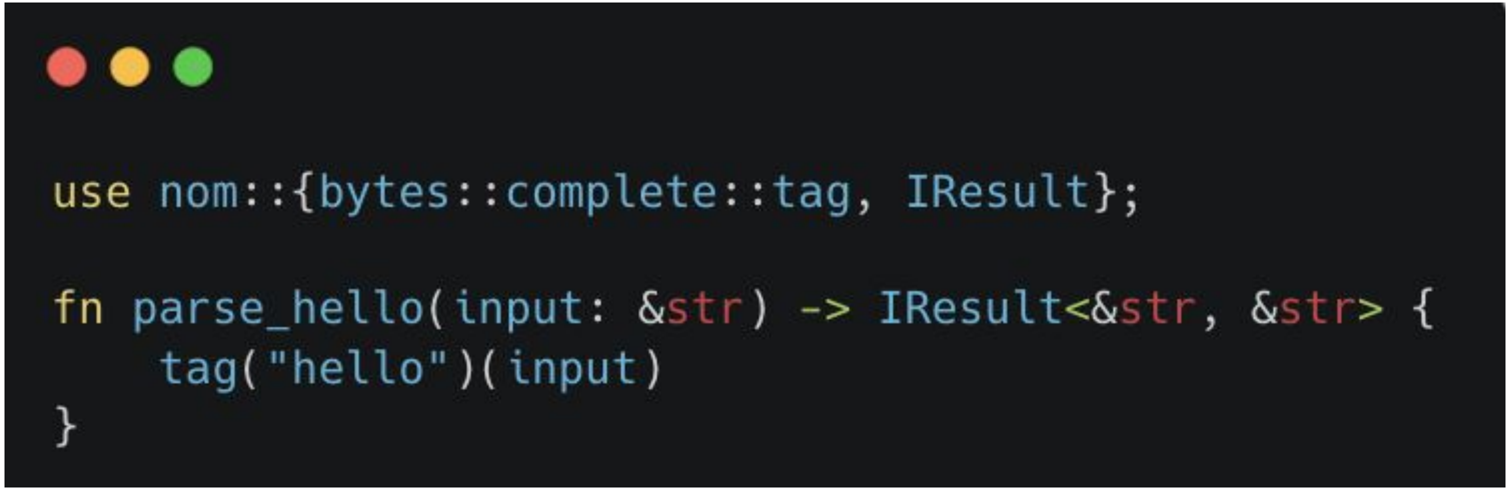
Success: parse the word “Hello”



```rust
fn parse_hello(input: &str) -> IResult<&str, &str, ()> {
  match input.strip_prefix("Hello") {
    Some(tail) => Ok((tail, "Hello")),
    None => Err(nom::Err(())),
  }
}
```

Nom parsers provided

Same Hello parser written using tag combinator parser



```rust
use nom::{bytes::complete::tag, IResult};

fn parse_hello(input: &str) -> IResult<&str, &str> {
    tag("hello")(input)
}
```

Built-in performance optimisations

Zero-copy parsing and streaming

- The data is not copied during parsing, leading to significant performance improvements

Why use nom?

Applications of nom

- Parsing binary formats
- Parsing text-based protocols
- Parsing configuration files

Resources for learning more about nom and parsing in Rust

- https://crates.io/crates/nom#why-use-nom