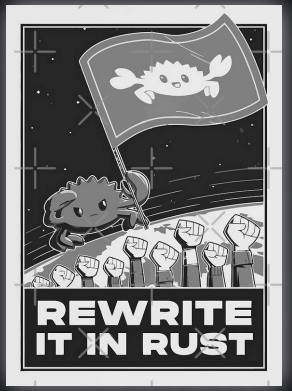
Crear un parser de **JSON** Con **LALRPOP** en **Rust**



LAMBDA CLASS

Que es LALRPOP?

look-ahead left-to-right (K = tokens to look ahead)

- LR(1) Parser Generator
- A partir de un grammar genera un módulo de Rust con una finite state machine para parsear el input.
- Puedes usar el builtin lexer o una librería como **logos**.
- Macros
- Similar a YACC, ANTLR, Menhir.

```
// grammar.lalrpop
use std::str::FromStr;

grammar;

pub Expr: i32 = {
    <l:Expr> "+" <r:Factor> => l + r,
    <l:Expr> "-" <r:Factor> => l - r,
    Factor,
};

Factor: i32 = {
    <l:Factor> "*" <r:Term> => l * r,
    <l:Factor> "/" <r:Term> => l / r,
    Term,
};

Term: i32 = {
    Num,
    "(" <Expr> ")",
};

Num: i32 = {
    r"[0-9]+" => i32::from_str(<>).unwrap(),
};
```

LAMBDA CLASS

Cómo funciona un LR(1) parser

Se compone de:

- Terminals: Tokens que da el lexer
- Non Terminals: Reglas definidas en el grammar.

Run Time

- Linear: escala con la longitud del input.

Parser actions:

- Shift step: Avanza el input stream por un simbolo. El token avanzado se convierte en un nodo en el parse tree.

- Reduce step: Aplica una regla de la grammar sobre los símbolos recientemente shifteados, uniendolos en un solo árbol con una nueva raíz

No backtracking

- Mirando K tokens adelante (en este caso 1)

EI AST

```rust
#[derive(Debug, Clone)]
pub struct Object<'input> {
    pub values: HashMap<&'input str, Value<'input>>,
    pub span: Span,
}

#[derive(Debug, Clone)]
pub struct Array<'input> {
    pub values: Vec<Value<'input>>,
    pub span: Span,
}

#[derive(Debug, Clone)]
pub enum Value<'input> {
    String(&'input str, Span),
    Number(&'input str, Span),
    Bool(bool, Span),
    Array(Array<'input>),
    Object(Object<'input>),
    Null(Span),
}
```

LAMBDA CLASS

Tokens

```
#[derive(Logos, Debug, PartialEq, Clone)]
#[logos(error = LexingError, skip r"[ \r\t\n\f]+")]
pub enum Token<'input> {
    #[regex(r"-?(?:0|[1-9]\d*)(?:\.\d+)?(?:[eE][+-]?\d+)?" )]
    Number(&'input str),
    #[regex(r#""([
-!#-\[\]-\x{10ffff}]|([\\](["\\/bfnrt]|[u][[:xdigit:]][[:xdigit:]][[:xdigit:]][[:xdigit:]]))*""#    )]
    String(&'input str),
    #[token("true")]
    True,
    #[token("false")]
    False,
    #[token("null")]
    Null,
    #[token("{")]
    LeftBrace,
    #[token("}")]
    RightBrace,
    #[token("[")]
    LeftBracket,
    #[token("]")]
    RightBracket,
    #[token(":")]
    Colon,
    #[token(",")]
    Comma,
}
```

LAMBDA CLASS

Grammar

```
grammar<'input>;

extern {
 type Location = usize;
 type Error = LexicalError;

 enum Token<'input> {
   // terminals
   "true" => Token::True,
   "false" => Token::False,
   ":" => Token::Colon,
   "," => Token::Comma,
   "null" => Token::Null,
   "{" => Token::LeftBrace,
   "}" => Token::RightBrace,
   "[" => Token::LeftBracket,
   "]" => Token::RightBracket,
   "number" => Token::Number(<&'input str>),
   "string" => Token::String(<&'input str>),
 }
}
```

JSON: Value

```
pub Value: ast::Value<'input> = {
    <lo:@L> <value:"string"> <hi:@R> => ast::Value::String(value, Span::new(lo, hi)),
    <lo:@L> <value:"number"> <hi:@R> => ast::Value::Number(value, Span::new(lo, hi)),
    <value:Array> => ast::Value::Array(value),
    <value:Object> => ast::Value::Object(value),
    <lo:@L> <value:"true"> <hi:@R> => ast::Value::Bool(true, Span::new(lo, hi)),
    <lo:@L> <value:"false"> <hi:@R> => ast::Value::Bool(false, Span::new(lo, hi)),
    <lo:@L> <value:"null"> <hi:@R> => ast::Value::Null(Span::new(lo, hi)),
}
```

JSON: Array

```
Comma<T>: Vec<T> = {
    <mut v:(<T> ",")*> <e:T?> => match e {
        None => v,
        Some(e) => {
            v.push(e);
            v
        }
    }
};

pub Array: ast::Array<'input> = {
    <lo:@L> "[" <values:Comma<Value>> "]" <hi:@R> => ast::Array {
        values,
        span: Span::new(lo, hi)
    }
}
```

JSON: Object

```
pub Object: ast::Object<'input> = {
    <lo:@L> "{" <values:Comma< (<"string"> ":" <Value>) >> "}" <hi:@R> => ast::Object {
        values: values.into_iter().map(|x| (x.0, x.1)).collect(),
        span: Span::new(lo, hi),
    }
}
```

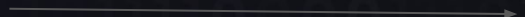
El Resultado

```rust
use lalrpop_util::lalrpop_mod;

use crate::lexer::Lexer;

mod ast;
mod lexer;
mod tokens;
lalrpop_mod!(pub grammar);

fn main() {
    let lexer = Lexer::new(r#"{ "hello": "world", "a":
[2, "s"] }"#);
    let parser = grammar::ObjectParser::new();
    dbg!(parser.parse(lexer).unwrap());
}
```



```
[src/main.rs:13] parser.parse(lexer).unwrap() = Object {
    values: {
        "\"hello\"": String(
            "\"world\"",
            Span {
                lo: 11,
                hi: 18,
            },
        ),
        "\"a\"": Array(
            Array {
                values: [
                    Number(
                        "2",
                        Span {
                            lo: 26,
                            hi: 27,
                        },
                    ),
                    String(
                        "\"s\"",
                        Span {
                            lo: 29,
                            hi: 32,
                        },
                    ),
                ],
                span: Span {
                    lo: 25,
                    hi: 33,
                },
            },
        ),
    },
    span: Span {
        lo: 0,
        hi: 35,
    },
}
```

LAMBDA CLASS

Code

github.com/edg-l/lalrpop-json