Лабораторная работа № 1.2. «Лексический анализатор на основе регулярных выражений»

28 февраля 2024 г.

Кирилл Киселев, ИУ9-61Б

Цель работы

Целью данной работы является приобретение навыка разработки простейших лексических анализаторов, работающих на основе поиска в тексте по образцу, заданному регулярным выражением.

Индивидуальный вариант

- Химические вещества: последовательности латинских букв и цифр, начинающиеся с заглавной буквы, при этом после цифры не может следовать строчная буква (атрибут: строка). Примеры: «CuSO4», «CH3CH2OH», «Fe2O3».
- Коэффициенты: последовательности десятичных цифр. Между коэффициентом и веществом пробел может отсутствовать.
- Операторы: «+», «->».

Реализация

main.rs

```rust
pub mod token;
pub mod tokenizer;
use std::env::args;
use std::fs::File;
use std::io::Read;

use crate::tokenizer::TokenizerErr;

fn main() {
    let args: Vec<String> = args().collect();
```

```
    if args.len() != 2 {
        println!("usage: cargo run <path_to_file>");
        return;
    }

    let file_name = &args[1];

    let mut file = File::open(file_name).unwrap();

    let mut raw_text = String::new();
    let _ = file.read_to_string(&mut raw_text);

    let mut tokenizer = tokenizer::Tokenizer::new(&raw_text);

    loop {
        let tmp = tokenizer.next();
        match tmp {
            Ok(token) => println!("{}", token),
            Err(err) => {
                println!("{}", err);

                if let TokenizerErr::EndOfStream() = err {
                    return;
                }
            }
        }
    }
}
```

token.rs

```
use std::fmt::{self, Display};

#[derive(Debug)]
pub enum TokenKind {
    Coefficient,
    Operator,
    Substance,
}

impl fmt::Display for TokenKind {
    fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
        match self {
            TokenKind::Coefficient => write!(f, "COEFFICIENT"),
            TokenKind::Operator => write!(f, "OPERATOR"),
            TokenKind::Substance => write!(f, "SUBSTANCE"),
```

```
        }
    }
}

#[derive(Debug)]
pub struct Token {
    val: String,
    kind: TokenKind,
    pos: (usize, usize),
}

impl Token {
    pub fn new(val: &str, kind: TokenKind, pos: (usize, usize)) -> Self {
        Self {
            val: val.to_string(),
            kind,
            pos,
        }
    }

    pub fn val(&self) -> &str {
        self.val.as_ref()
    }

    pub fn kind(&self) -> &TokenKind {
        &self.kind
    }

    pub fn pos(&self) -> (usize, usize) {
        self.pos
    }
}

impl Display for Token {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}",
            format!(
                "{} ({}, {}): {}",
                self.kind(),
                self.pos().0 + 1,
                self.pos().1 + 1,
                self.val()
            )
        )
```

```
    }
}
```

tokenizer.rs

```rust
use std::usize;

use crate::token::{self, Token};

#[derive(Debug)]
pub enum TokenizerErr {
    SyntaxErr((usize, usize)),
    EndOfStream(),
}

impl std::fmt::Display for TokenizerErr {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            TokenizerErr::SyntaxErr(pos) => {
                write!(f, "Syntax error ({}, {})", pos.0 + 1, pos.1 + 1)
            }
            TokenizerErr::EndOfStream() => write!(f, "End of stream"),
        }
    }
}

#[derive(Debug)]
pub struct Tokenizer {
    raw_text: Vec<String>,
    pos: (usize, usize),
    coeff_regex: fancy_regex::Regex,
    op_regex: fancy_regex::Regex,
    substance_regex: fancy_regex::Regex,
}

impl Tokenizer {
    pub fn new(text: &str) -> Self {
        let tmp = Self {
            raw_text: text.lines().map(str::to_string).collect(),
            pos: (0, 0),
            coeff_regex: fancy_regex::Regex::new(r"(?<coeff>[1-9]\d*)").unwrap(),
            op_regex: fancy_regex::Regex::new(r"(?<op>\+|->)").unwrap(),
            substance_regex: fancy_regex::Regex::new(r"(?<substance>([A-Z][a-z]?([1-9]\d*)?)
                .unwrap(),
        };
```

```
    tmp
}

pub fn next(&mut self) -> Result<token::Token, TokenizerErr> {
    self.skip_whitespaces();

    if self.pos.0 >= self.raw_text.len() {
        return Err(TokenizerErr::EndOfStream());
    }

    let coeff_captures = self
        .coeff_regex
        .captures_from_pos(&self.raw_text[self.pos.0], self.pos.1)
        .unwrap();
    let substance_captures = self
        .substance_regex
        .captures_from_pos(&self.raw_text[self.pos.0], self.pos.1)
        .unwrap();
    let op_captures = self
        .op_regex
        .captures_from_pos(&self.raw_text[self.pos.0], self.pos.1)
        .unwrap();

    let mut coeff_pos = (usize::MAX, usize::MAX);
    let mut coeff_value = "";

    let mut op_pos = (usize::MAX, usize::MAX);
    let mut op_value = "";

    let mut substance_pos = (usize::MAX, usize::MAX);
    let mut substance_value = "";

    if let Some(captures) = coeff_captures {
        coeff_pos.0 = captures.get(1).unwrap().start();
        coeff_pos.1 = captures.get(1).unwrap().end();
        coeff_value = captures.get(1).unwrap().as_str();
    }

    if let Some(captures) = substance_captures {
        substance_pos.0 = captures.get(1).unwrap().start();
        substance_pos.1 = captures.get(1).unwrap().end();
        substance_value = captures.get(1).unwrap().as_str();
    }

    if let Some(captures) = op_captures {
        op_pos.0 = captures.get(1).unwrap().start();
```

```
            op_pos.1 = captures.get(1).unwrap().end();
            op_value = captures.get(1).unwrap().as_str();
        }

        let start = coeff_pos.0.min(op_pos.0).min(substance_pos.0);

        if start != self.pos.1 {
            let line = self.pos.0;
            let col = self.pos.1;
            self.skip_err();
            return Err(TokenizerErr::SyntaxErr((line, col)));
        }

        let mut kind = token::TokenKind::Coefficient;
        let mut value = coeff_value;
        let mut end = coeff_pos.1;

        if start == op_pos.0 {
            kind = token::TokenKind::Operator;
            end = op_pos.1;
            value = op_value;
        } else if start == substance_pos.0 {
            kind = token::TokenKind::Substance;
            end = substance_pos.1;
            value = substance_value;
        }

        self.pos.1 = end;

        Ok(Token::new(value, kind, (self.pos.0, start)))
    }

    fn skip_err(&mut self) {
        loop {
            self.skip_whitespaces();

            if self.pos.0 >= self.raw_text.len() {
                return;
            }

            let mut idx = usize::MAX;

            if let Ok(Some(capture)) = self
                .op_regex
                .captures_from_pos(&self.raw_text[self.pos.0], self.pos.1)
            {
```

```
            idx = idx.min(capture.get(1).unwrap().start());
        }

        if let Ok(Some(capture)) = self
            .substance_regex
            .captures_from_pos(&self.raw_text[self.pos.0], self.pos.1)
        {
            idx = idx.min(capture.get(1).unwrap().start())
        }

        if let Ok(Some(capture)) = self
            .coeff_regex
            .captures_from_pos(&self.raw_text[self.pos.0], self.pos.1)
        {
            idx = idx.min(capture.get(1).unwrap().start())
        }

        if idx < usize::MAX {
            self.pos.1 = idx;
            return;
        } else {
            self.pos.0 += 1;
            self.pos.1 = 0;
        }
    }
}

fn skip_whitespaces(&mut self) {
    if self.pos.0 >= self.raw_text.len() {
        return;
    }

    let mut counter = self.pos.1;
    for c in self.raw_text[self.pos.0].chars().skip(self.pos.1) {
        if !c.is_whitespace() {
            self.pos.1 = counter;
            return;
        }
        counter += 1;
    }

    if counter == self.raw_text[self.pos.0].len() {
        self.pos.0 += 1;
        self.pos.1 = 0;
        self.skip_whitespaces();
    }
```

```
    }
}
```

Тестирование

Входные данные

```
  2 eC Fe2O3 + 6C -> 2 Fe --
  2H2 + O2 -> H2O
```

Вывод на `stdout`

```
COEFFICIENT (1, 3): 2
Syntax error (1, 5)
SUBSTANCE (1, 6): C
SUBSTANCE (1, 8): Fe2O3
OPERATOR (1, 14): +
COEFFICIENT (1, 16): 6
SUBSTANCE (1, 17): C
OPERATOR (1, 19): ->
COEFFICIENT (1, 22): 2
SUBSTANCE (1, 24): Fe
Syntax error (1, 27)
COEFFICIENT (2, 3): 2
SUBSTANCE (2, 4): H2
OPERATOR (2, 7): +
SUBSTANCE (2, 9): O2
OPERATOR (2, 12): ->
SUBSTANCE (2, 15): H2O
End of stream
```

Вывод

В ходе выполнения данной лабораторной работы были получены навыки по разработке лексического анализатора работающего на основе сопоставления с образцом. Для реализации анализатора была изучена библиотека *fancy_regex*, которая предоставляет возможность работы с регулярными выражениями в языке Rust.