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

26 февраля 2024 г.

Кабанов Андрей Юрьевич, ИУ9-62Б

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

- Регулярные выражения: ограничены знаками «/», не могут пересекать границы текста, содержат escape-последовательности «\n», «/», «\».
- Строковые литералы: ограничены тремя кавычками («”””»), могут занимать несколько строчек текста, не могут содержать внутри более двух кавычек подряд.

Реализация

```
import re
class Token:
    token_type = ""
    data = ""
    coords = None
    def __init__(self, token_type, data, coords):
        self.token_type = token_type
        self.data = data
        self.coords = coords
    def __str__(self):
        return f"Token(type = {self.token_type},
        data = {self.data}, coords = ({self.coords[0]+1}, {self.coords[1]+1}))"
def lex(lines):
    reg_reg = r"^/(?:\\/|\\n|\\\\|[^\n\r])*/$"
```

```
reg_str_start_finish = r"^\"\"\"(?:[^\"]|\"(?!\"))*\"\"\"$"
reg = r"(?P<regex>/(?:\\/|\\n|\\\\|[^\n\r])*/)|\
(?P<str>\"\"\"(?:[^\"]|\"(?!\"))*\"\"\")|(?P<space>\s+)"
text = ""
lines_len = []
if len(lines) > 1:
    text = lines[0]
    lines_len.append(len(lines[0]))
    for line in lines[1:]:
        text = text + "\n" + line
        lines_len.append(len(line))
else:
    text = lines[0]
    lines_len.append(len(lines[0]))
line_num = 0
current_len = 0
past_coords = (0, 0)
for i in re.finditer(reg, text):
    data = i.group("str")
    x, y = i.span()
    if x > current_len+lines_len[line_num]+line_num:
            current_len+=lines_len[line_num]
            line_num+=1
            while x > current_len+lines_len[line_num]+line_num:
                current_len+=lines_len[line_num]
                line_num+=1
    col = x - current_len - line_num
    end_col = col + (y-x)
    end_line = line_num
    while end_col > lines_len[end_line]:
        end_col -= lines_len[end_line] + 1
        end_line+=1
    if data is not None:
        if past_coords and past_coords != (line_num, col):
            yield Token("error", "", past_coords)
        yield Token("str", data, (line_num, col))
    data = i.group("regex")
    if data is not None:
        if past_coords and past_coords != (line_num, col):
            yield Token("error", "", past_coords)
        yield Token("regex", data, (line_num, col))
    data = i.group("space")
    if data is not None:
        if past_coords and past_coords != (line_num, col):
            yield Token("error", "", past_coords)
    past_coords = (end_line, end_col)
```

```
    if past_coords[0] != len(lines_len)-1 or \
     past_coords[1] != lines_len[past_coords[0]]:
        yield Token("error", "", past_coords)

lines = []
with open(r"test.txt", "r") as f:
    lines = f.readlines()

for token in lex(lines):
    print(token)
```

Тестирование

Входные данные

```
"""ewjfejfewkfk2\n
"""/jefjfjjef/

"""aa"""  @ /a\\\/\naa/ 2
```

Вывод на `stdout`

```
Token(type = error,          data = , coords = (1, 1))
Token(type = error,          data = , coords = (2, 1))
Token(type = regex,          data = /jefjfjjef/, coords = (2, 4))
Token(type = error,          data = , coords = (4, 1))
Token(type = error,          data = , coords = (4, 11))
Token(type = regex,          data = /a\\\/\naa/, coords = (4, 13))
Token(type = error,          data = , coords = (4, 25))
```

Вывод

При выполнении данной лабораторной работы были приобретены навыки разработки лексического анализатора на основе регулярных выражений. А также были изучены сами регулярные выражения.