

گزارش فاز دوم پروژه درس طراحی کامپایلر:

:PLY

PLY یک ابزار مناسب برای پیاده سازی بخش مهم کامپایلر lex و yacc به زبان پایتون می باشد. PLY شامل دو ماژول مجزا lex.py و yacc.py است که هر دو در پکیج ply موجود هستند. در این فاز ما میخواهیم از ماژول lex.py استفاده نماییم و در ادامه نحوه عملکرد این ماژول را شرح میدهیم.

:LEX

Lex.py در شکستن برنامه ورودی به مجموعه توکن های تعریف شده به صورت عبارت منظم مورد استفاده قرار میگیرد. به این صورت که کاراکتر های برنامه ورودی را بررسی می نماید و با توکن های معرفی شده به صورت عبارت منظم مقایسه مینماید در صورتی که کاراکتر ها به گونه ای چیده شده و با یکی از توکن ها مطابقت داشته باشند آنگاه آن « دسته کاراکتر را از برنامه ورودی جدا نموده و در گروه توکن مورد نظر قرار داده و به صورت خروجی برمی گرداند و سپس به سراغ کاراکتر های بعدی رفته و اینگونه تمام برنامه ورودی را به صورت توکن های تعریف شده شسسته و تقسیم بندی مینماید.

:Token List

همانگونه که در بالا اشاره نمودیم، ابزار Lex شامل یک بخش ضروری به نام token list می باشد که شامل لیست تمام توکن هایی است که ممکن است در برنامه شناسایی شوند.

:Specification of tokens

با اعلام لیست توکن ها می بایست هر توکن را جداگانه با استفاده از قوانین عبارت های منظم تعریف نماییم که این عمل توسط ماژول re در پایتون انجام می شود. این شناسایی به دو صورت انجام میشود: اگر توکن مورد نظر ساده باشد کافیت تعریف قانون مربوط به آن توکن به صورت یک string انجام شود. در غیر اینصورت اگر نیاز به انجام عملیات های مختلف در حین تعریف توکن باشد باید تعریف توکن مورد نظر را به صورت یک تابع پیاده سازی کنیم.

:Line numbers and positional information

به صورت پیش فرض ابزار lex.py توانایی شمارش خط را ندارد برای این عمل تابعی تحت عنوان t_newline میسازیم که لکسر با خواندن کاراکتر \n توکن newline را شناسایی نموده و به متغیر lineno یک واحد میافزاید. این تابع مقداری را بازنمیگرداند.

:Ignored characters

قانون t_ignore برای lex.py رزرو شده است تا کاراکتر هایی که باید در برنامه ورودی نادیده گرفته شوند را مورد بررسی قرار ندهد مانند فاصله ها و tabها

:Building and using the lexer

در نهایت برای استفاده از یک لکسر و مشخص کردن توکن های فایل ورودی ابتدا با استفاده از تابع lex.lex() یک لکسر ایجاد میکنیم مثل: Lexer = lex.lex(). پس از ایجاد لکسر مورد نظر میتوانیم از دو متد lexer.input(data) برای دادن فایل برای مشخص نمودن توکن های آن و همچنین lexer.token() برای بازگرداندن توکن یافت شده از فایل ورودی، استفاده نماییم.

برنامه های تست شده به همراه خروجی آن ها:

The screenshot shows the PyCharm IDE interface. The main editor window displays a Python script for a lexer. The script defines a list of tokens and a function to build the lexer. The output window shows the execution results of the lexer for various input strings.

```
# Build the lexer
lexer = lex.lex()

data = '''
while(1){
    for(u++12
    int m==6;
}

'''

# Give the lexer some input
lexer.input(data)
```

The output window shows the following results:

```
LexToken(EOF, 'for', 3, 17)
LexToken(PLUSPLUS, '++', 3, 22)
LexToken(NUMBER, 12, 3, 24)
LexToken(INT, 'int', 4, 28)
LexToken(IDENTIFIER, 'm', 4, 32)
LexToken(ASSIGN, '==', 4, 34)
LexToken(NUMBER, 6, 4, 36)
LexToken(OTHEROP, '{', 4, 37)
LexToken(RBRACKET, '}', 4, 42)
Process finished with exit code 0
```

برنامه های تست شده به همراه خروجی آن ها:

The screenshot shows an IDE window titled "Test" with a menu bar (File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help). The main editor area displays the output of a Python script, showing a list of tokens and their corresponding line and column numbers. The tokens are listed in a single column, with line numbers ranging from 67 to 78. The tokens include various identifiers, keywords, and operators, such as "LexToken(FOR, 'for', 2, 2)", "LexToken(PAREN, '(', 2, 6)", "LexToken(INT, 'int', 2, 6)", "LexToken(IDENTIFIER, 'i', 2, 10)", "LexToken(ASSIGN, '=', 2, 11)", "LexToken(LNUMBER, 0, 2, 12)", "LexToken(OTHEROP, ';', 2, 13)", "LexToken(IDENTIFIER, 'i', 2, 15)", "LexToken(OTHEROP, '<', 2, 16)", "LexToken(ASSIGN, '=', 2, 17)", "LexToken(IDENTIFIER, 'n', 2, 18)", "LexToken(OTHEROP, ';', 2, 20)", "LexToken(IDENTIFIER, 'i', 2, 22)", "LexToken(PUSPLUS, '++', 2, 23)", "LexToken(PAREN, ')', 2, 25)", "LexToken(BRACKET, '{', 3, 28)", "LexToken(IF, 'if', 4, 34)", "LexToken(IDENTIFIER, 'f', 4, 36)", "LexToken(PAREN, '(', 4, 37)", "LexToken(IDENTIFIER, 'i', 4, 38)", "LexToken(OTHEROP, '>', 4, 39)", "LexToken(IDENTIFIER, 'b', 4, 40)", "LexToken(PAREN, ')', 4, 41)", "LexToken(BRACKET, '{', 5, 47)", "LexToken(IDENTIFIER, 'b', 6, 57)", "LexToken(ASSIGN, '=', 6, 58)", "LexToken(IDENTIFIER, 'h', 6, 59)", "LexToken(PLUS, '+', 6, 60)", "LexToken(IDENTIFIER, 'i', 6, 61)", "LexToken(OTHEROP, ';', 6, 62)".

The left sidebar shows the project structure, including a "Test" folder and a "Test" file. The "Test" file is selected, and its contents are displayed in the main editor. The code in the "Test" file is a Python script that defines a lexer and tests it with various inputs. The script includes a function "lex" that takes a string and returns a list of tokens. The function uses a dictionary "tokens" to map strings to token types. The function also includes a loop that iterates over the input string and calls the "lex" function for each character. The output of the function is printed to the console.

The bottom status bar shows the file encoding as UTF-8, 4 spaces, Python 3.8, and the date/time as 11:40 AM, 11/1/2019.

برنامه های تست شده به همراه خروجی آن ها:

```
1: Project
Test
File Edit View Language Code Refactor Run Tools Windows Help
Test (C:\Users\Saleh\Downloads\Compressed\Compiler\test) - \Test.py - Python
C:\Python\python.exe C:/Users/Saleh/Downloads/Compressed/Compiler/Test/Test.py
C:\Python\python.exe C:/Users/Saleh/Downloads/Compressed/Compiler/Test/Test.py
LexToken(while, 'while', 2, 2)
LexToken(OTHEROP, '[', 2, 7)
LexToken(OTHEROP, ']', 2, 8)
LexToken(IDENTIFIER, 'if', 3, 2, 10)
LexToken(ELSECLAUSE, '@', 2, 15)
LexToken(IDENTIFIER, ']', 2, 17)
LexToken(PUSPLUS, '++', 2, 18)
LexToken(OTHEROP, ']', 2, 20)
LexToken(IDENTIFIER, 'for', 4, 3, 26)
LexToken(OTHEROP, '>', 3, 30)
LexToken(IDENTIFIER, 'while', 3, 31)
LexToken(ELSECLAUSE, '}', 4, 38)
Process finished with exit code 0

t.lexer.skip(1)
# Build the Lexer
lexer = lex.Lex()

data = ...
while() if 3 0 ++]
for while
}
# Give the Lexer some input
lexer.input(data)

# Tokenize
while True:
    tok = lexer.token()
    if not tok:
        break # No more input
```