Создание программы лексического анализатора (лексера)

Цель реализовать программу лексического анализатора.
Для создания программы мы пользуемся готовыми инструментами программа генерации лексического анализатора flex и библиотеками Питона PLY (https://www.dabeaz.com/ply/)

Сканер преобразует исходный файл исходной программы в серию токенов, содержащих информацию, которая будет использоваться на более поздних этапах компиляции.

Сканер должен обрабатывать преобразования из литералов с
целыми и
действительными значениями в целые и числовые данные с действительными значениями.

То есть, последовательность символов 3.1415E+3, при сканировании необходимо преобразовать в double с этим значением.
Целое число 137, необходимо преобразовать в целое число.

Можно предположить, что любой числовой литерал, целочисленный или действительный, может быть преобразован в соответствующий тип без ошибок, и поэтому можно не беспокоиться о литералах, которые переполняются или теряют значение.

Наш ЯП использует комментарии, такие же, как в C++.
• Однострочный комментарий начинается с // и продолжается до конца строки.
• Многострочные комментарии начинаются с /* и заканчиваются первым последующим */.

В комментарии допускается любой символ, кроме последовательности */, которая завершает текущий комментарий.

Многострочные комментарии не вкладывают друг в друга.

Сканер должен принимать любые комментарии из входного потока и игнорировать их.

Если файл заканчивается незавершенным комментарием, сканер должен сообщить об ошибке.

Наш ЯП использует следующие операторы и знаки препинания:
+ - * / % < <= > >= = == != && || ! ; , . [ ] ( ) { } []

Обратите внимание, что [,] и [] — это три разных токена, и что для оператора [] (используемого для объявления переменной типа массива), а также для других двух-символьных операторов между двумя символами не должно быть пробела.
Каждый одно-символьный оператор имеет тип маркера, равный его значению ASCII, в то время как много-символьные операторы имеют связанные с ними именованные типы маркеров. Например, для
[] - тип токена T_Dims,
|| - тип токена T_Or.

Практические шаги

1. разбираем пример calk.py, и пример с Хабра («Разбор кода и построение синтаксических деревьев с PLY. Основы _ Хабр.pdf»). Из архива «habr-ply-examples.zip» извлекаем в рабочую папку файлы и запускаем в Питоне.

2. изменяем пример оставляем только лексический разбор.

```
# -----------------------------------------------------------------------------
# calc.py
#
# A simple calculator with variables -- all in one file.
# -----------------------------------------------------------------------------

tokens = (
    'NAME','NUMBER',
    'PLUS','MINUS','TIMES','DIVIDE','EQUALS',
    'LPAREN','RPAREN',
    )

# Tokens

t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES   = r'\*'
t_DIVIDE  = r'/'
t_EQUALS  = r'='
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
t_NAME    = r'[a-zA-Z_][a-zA-Z0-9_]*'

def t_NUMBER(t):
    r'\d+'
    try:
        t.value = int(t.value)
    except ValueError:
        print("Integer value too large %d", t.value)
        t.value = 0
    return t

# Ignored characters
t_ignore = " \t"

def t_newline(t):
    r'\n+'
    t.lexer.lineno += t.value.count("\n")

def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)

# Build the lexer
```

```
import ply.lex as lex
lexer = lex.lex()

if __name__=="__main__":
    data = ' 123 + 345 * ( 45 – 33) / 2334 '
    lexer.input(data)

    while True:
        tok = lexer.token() # читаем следующий токен
        if not tok: break      # закончились печеньки
        print ( tok )
```

Данный пример выводит в консоль содержимого потока переменной data.

3. Добавляем или изменяем недостающие токены согласно спецификации. Требования к лексическому анализатору находятся в файле «Scanner_requrements.txt».

Тестирование

4. Добавляем возможность считывать data из заданного файла.

5. Сравниваем содержимое файла с тем же именем расширением .out
Делаем сообщение если все ОК и файлы идентичны. Иначе выводим инфо о различиях.

Для тестирование лексера, распаковываем архив с тестами «lexer_tests.zip» в рабочую папку где находится наш лексер. В папке «lexer_tests» содержатся различные входные файлы с раширением .frag (содержат данные для лексического анализа) и файлы с расширением .out. (представляют ожидаемый результат).

- Необходимо сравнить вывод программы с тем, что представлен в файле .out в качестве первого шага в тестировании.
- Затем просмотреть тестовые файлы и подумайте, какие случаи не охвачены.
- Создайте несколько собственные тестовые файлы для проверки сканера.

Какие лексемы похожи на числа, но не являются ими?

Какие последовательности могут запутать вашу обработку комментариев?

Важно проверить разные варианты ввода, и убедиться, что лексер может обрабатывать все варианты, правильно разбивая входную последовательность на токены, либо сообщая о какой-то ошибке, если этого невозможно сделать.

Обратите внимание, что лексический анализ отвечает только за правильное разбиение входного потока и категоризацию каждой лексемы по типу.

Сканер будет принимать синтаксически фиктивные последовательности, такие как:
int if array + 4.5 [bool}