2025_2 - COMPILADORES - METATURMA

Descrição Visualizar envios

AV1 - Análise Léxica

Data de entrega: sexta, 29 Ago 2025, 23:59
Arquivos requeridos: driver.py, Lexer.py (Baixar)
Tipo de trabalho: Trabalho individual

O objetivo deste trabalho é implementar um analisador léxico para um subconjunto da linguagem cool, que descreve expressões aritméticas (vide Seção 7.12 do manual da linguagem). Seu analisador léxico deverá reconhecer os seguintes tokens:

1. EOF: fim de arquivo
2. NLN: quebra de linha
3. WSP: espaço em branco
4. COM: comentário
5. EQL: sinal de igual (=)
6. ADD: sinal de adição (+)
7. SUB: sinal de subtração (-)
8. MUL: sinal de multiplicação (*)
9. DIV: sinal de divisão (/)
10. LEQ: sinal de menor ou igual (<=)
11. LTH: sinal de menor que (<)
12. NEG: sinal de menos unário
13. NOT: negação booleana (not)
14. LPR: parênteses esquerdo
15. RPR: parênteses direito
16. TRU: a constante booleana "true"
17. FLS: a constante booleana "false"
18. INT: números inteiros 0 | (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*
19. BIN: números binários 0(b|B)(0|1)*
20. OCT: números octais 0(0|1|2|3|4|5|6|7|8|9)+
21. HEX: números hexadecimais 0(x|X)(0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F)*

Comentários podem ser de dois tipos:

- Comentário de linha, que começa com um duplo sinal de menos (--). Um comentário de linha faz com que o resto da linha seja ignorada.
- Comentário de bloco, que contém texto entre (* e *). Todo o texto entre esses marcadores deve ser ignorado.

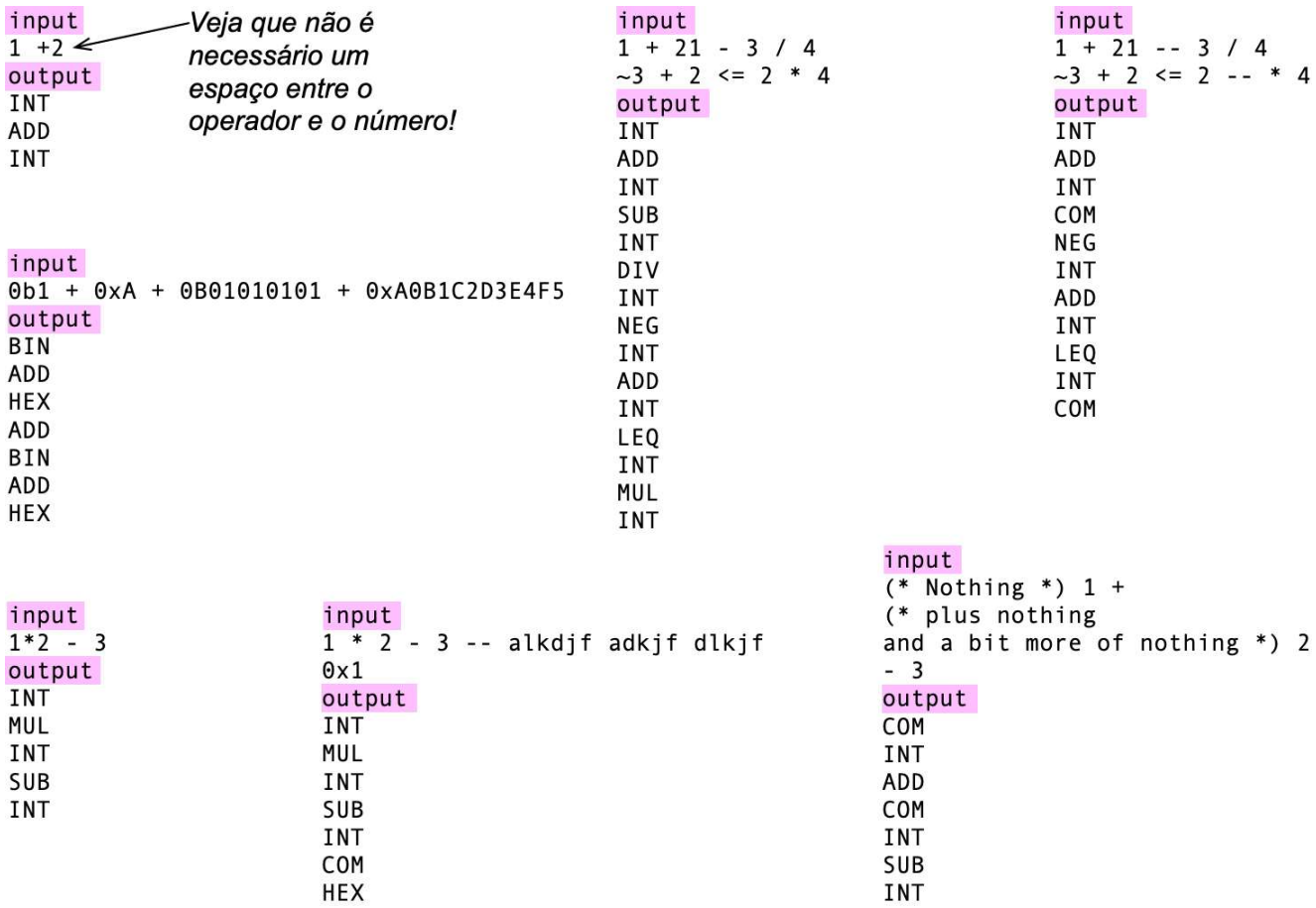
Uma parte da implementação já está pronta para você. Você deverá reusar a definição de tokens (classe TokenType) e a implementação de Tokens (classe Token). Seu objetivo será implementar uma classe Lexer. Esta classe é um gerador de tokens. O método de geração de tokens segue abaixo:

```
    def tokens(self):
        token = self.getToken()
        while token.kind != TokenType.EOF:
            if token.kind != TokenType.WSP and token.kind != TokenType.NLN:
                yield token
            token = self.getToken()
```

Note que o gerador de tokens retorna qualquer tipo de token, exceto espaços em branco e comentários. Seu código será testado pelo driver abaixo:

```
if __name__ == "__main__":
    lexer = Lexer(sys.stdin.read())
    for token in lexer.tokens():
        print(f"{token.kind.name}")
```

Em outras palavras, os testes mostrados na figura abaixo deverão funcionar:



Parte do trabalho está implementada para você. Você deverá fazer o upload de dois arquivos: `Lexer.py` e `driver.py`. Você não precisa alterar nada em driver.py.

Arquivos requeridos

driver.py

```
1  import sys
2  from Lexer import *
3
4  if __name__ == "__main__":
5      """
6      Este arquivo nao deve ser alterado, mas deve ser enviado para resolver o
7      VPL. O arquivo contem o codigo que testa a implementacao do analisador
8      lexico.
9      """
10     lexer = Lexer(sys.stdin.read())
11     for token in lexer.tokens():
12         print(f"{token.kind.name}")
```

Lexer.py

```
import sys
import enum


class Token:
    """
    This class contains the definition of Tokens. A token has two fields: its
    text and its kind. The "kind" of a token is a constant that identifies it
    uniquely. See the TokenType to know the possible identifiers (if you want).
    You don't need to change this class.
    """
    def __init__(self, tokenText, tokenKind):
        # The token's actual text. Used for identifiers, strings, and numbers.
        self.text = tokenText
        # The TokenType that this token is classified as.
        self.kind = tokenKind


class TokenType(enum.Enum):
    """
    These are the possible tokens. You don't need to change this class at all.
    """
    EOF = -1  # End of file
    NLN = 0   # New line
    WSP = 1   # White Space
    COM = 2   # Comment
    STR = 3   # Strings
    TRU = 4   # The constant true
    FLS = 5   # The constant false
    INT = 6   # Number (integers)
    BIN = 7   # Number (binary)
    OCT = 8   # Number (octal)
    HEX = 9   # Number (hexadecimal)
    EQL = 201
    ADD = 202
    SUB = 203
    MUL = 204
    DIV = 205
    LEQ = 206
    LTH = 207
    NEG = 208
    NOT = 209
    LPR = 210
    RPR = 211


class Lexer:

    def __init__(self, source):
        """
        The constructor of the lexer. It receives the string that shall be
        scanned.
        TODO: You will need to implement this method.
        """
        pass

    def tokens(self):
        """
        This method is a token generator: it converts the string encapsulated
        into this object into a sequence of Tokens. Examples:

        >>> l = Lexer("10")
        >>> [tk.kind.name for tk in l.tokens()]
        ['INT']

        >>> l = Lexer("01")
        >>> [tk.kind.name for tk in l.tokens()]
        ['OCT']

        >>> l = Lexer("0b1")
        >>> [tk.kind.name for tk in l.tokens()]
        ['BIN']

        >>> l = Lexer("0B1")
        >>> [tk.kind.name for tk in l.tokens()]
        ['BIN']

        >>> l = Lexer("0x1")
        >>> [tk.kind.name for tk in l.tokens()]
        ['HEX']

        >>> l = Lexer("0X1")
        >>> [tk.kind.name for tk in l.tokens()]
        ['HEX']

        >>> l = Lexer("0X1 + 0xA + 0XABCDEF + 0xA0B1C2D3E4F5")
        >>> [tk.kind.name for tk in l.tokens()]
        ['HEX', 'ADD', 'HEX', 'ADD', 'HEX', 'ADD', 'HEX']

        >>> l = Lexer("0b1 + 0xA + 0B01010101 + 0xA0B1C2D3E4F5")
        >>> [tk.kind.name for tk in l.tokens()]
        ['BIN', 'ADD', 'HEX', 'ADD', 'BIN', 'ADD', 'HEX']

        >>> l = Lexer('1 * 2 - 3')
        >>> [tk.kind.name for tk in l.tokens()]
        ['INT', 'MUL', 'INT', 'SUB', 'INT']

        >>> l = Lexer('1 * 2 - 3 -- alkdjf adkjf dlkjf \\n')
        >>> [tk.kind.name for tk in l.tokens()]
        ['INT', 'MUL', 'INT', 'SUB', 'INT', 'COM']

        >>> l = Lexer('1 * 2 - 3 -- alkdjf adkjf dlkjf \\n0x23 + 012')
        >>> [tk.kind.name for tk in l.tokens()]
```

```
104         ['INT', 'MUL', 'INT', 'SUB', 'INT', 'COM', 'HEX', 'ADD', 'OCT']
105         """
106         token = self.getToken()
107         while token.kind != TokenType.EOF:
108             if token.kind != TokenType.WSP and token.kind != TokenType.NLN:
109                 yield token
110             token = self.getToken()
111 
112     def getToken(self):
113         """
114         Return the next token.
115         TODO: Implement this method!
116         """
117         token = None
118         return token
```

VPL