Universidade Federal de Santa Catarina

Relatório - Descrição do Analisador Léxico

INE5426 - Construção de Compiladores

Helena Kunz Aires (17100521)
Paola de Oliveira Abel (17100531)
Paulo Barbato Fogaça de Almeida (18100542)
Pedro Henrique Aquino Silva (18102719)

Florianópolis
2022

Sumário

1 Identificação dos Tokens

1.1 Tokens de palavras reservadas

Há 13 tokens destinados a palavras reservadas da linguagem definida pela gramática CC-2022-1. São eles: DEF, BREAK, READ, RETURN, IF, ELSE, FOR, NEW, NULL, PRINT, INT, FLOAT, e STRING.

1.2 Tokens de símbolos terminais triviais

Há 20 tokens destinados aos símbolos terminais triviais da gramática CC-2022-1. São eles: LPARENTESES, RPARENTESES, LBRACKET, RBRACKET, LCURLYBRACKET, RCURLYBRACKET, SEMICOLON, COMMA, ASSIGNMENT, EQUAL, MINUS, PLUS, LESSTHAN, GREATERTHAN, LESSEQUAL, GRATEREQUAL, DIFFERENT, DIV, MOD e MULT.

1.3 Tokens de símbolos terminais não-triviais

Há 4 tokens destinados aos símbolos terminais não-triviais (ou seja, uma palavra lexicalmente válida para a linguagem que não é especificamente definida na gramática e que, portanto, pode tomar um valor arbitrário, como por exemplo um número ou o nome de uma função) da gramática CC-2022-1. São eles: IDENT, INT_CONSTANT, FLOAT_CONSTANT e STRING_CONSTANT.

1.4 Tokens removíveis

Há 2 tokens cujo propósito é identificar caracteres e/ou sequências de caracteres que serão ignorados pela análise léxica. São eles: IGNORE e COMMENT.

2 Definições regulares

As definições regulares foram criadas dentro do código para o exercício utilizando os mecanismos da linguagem Python para expressões regulares. Escolhemos manter esta representação no relatório para reter consistência com o código apresentado como solução do exercício. Notamos que o Python permite o uso do caractere '\d' para identificação de dígitos [0-9] nas expressões regulares.

2.1 Definições regulares para palavras reservadas

Foi implementada uma única regra para identificar palavras reservadas. Um dicionário foi especificado, com cada chave equivalente a uma palavra reservada e o valor correspondente sendo o token. Essa regra é definida na função t_IDENT, que define certos símbolos terminais não-triviais (identificadores); tal função é apresentada na seção 2.3.

```
keywords = {
                "def": "DEF",
                "break": "BREAK",
                "read": "READ",
                "return": "RETURN",
                "if": "IF",
                "else": "ELSE",
                "for": "FOR",
                "new": "NEW",
                "null": "NULL",
                "print": "PRINT",
                "int": "INT",
                "float": "FLOAT",
                "string": "STRING",
}
```

2.2 Definições regulares para símbolos terminais triviais

```
t_LPARENTESES = r"\("
t_RPARENTESES = r"\)"
t_RPARENTESES = r"\)"
t_LBRACKET = r"\["
t_RBRACKET = r"\]"
t_LCURLYBRACKET = r"{"
t_RCURLYBRACKET = r"}"
```

```
    t_SEMICOLON = r";"
    t_COMMA = r"\,"
    t_ASSIGNMENT = r"="
    t_EQUAL = r"=="
    t_MINUS = r"-"
    t_PLUS = r"\+"
    t_LESSTHAN = r"<"
    t_GREATERTHAN = r">"
    t_LESSEQUAL = r"<="
    t_GREATEREQUAL = r">="
    t_DIFFERENT = r"!="
    t_DIV = r"/"
    t_MOD = r"%"
    t_MULT = r"\*"
```

2.3 Definições regulares para símbolos terminais não-triviais

```
def t_IDENT(self, t):
       r"[a-zA-Z][a-zA-Z0-9_]*"
       t.type = self.keywords.get(t.value, "IDENT")
       return t

def t_FLOAT_CONSTANT(self, t):
       r"\d+\.\d+(E[+-]?\d+)?"
       return t

def t_INT_CONSTANT(self, t):
       r"\d+"
       return t

def t_STRING_CONSTANT(self, t):
       r"\".*?\""
       return t
```

2.4 Definições regulares para caracteres removíveis

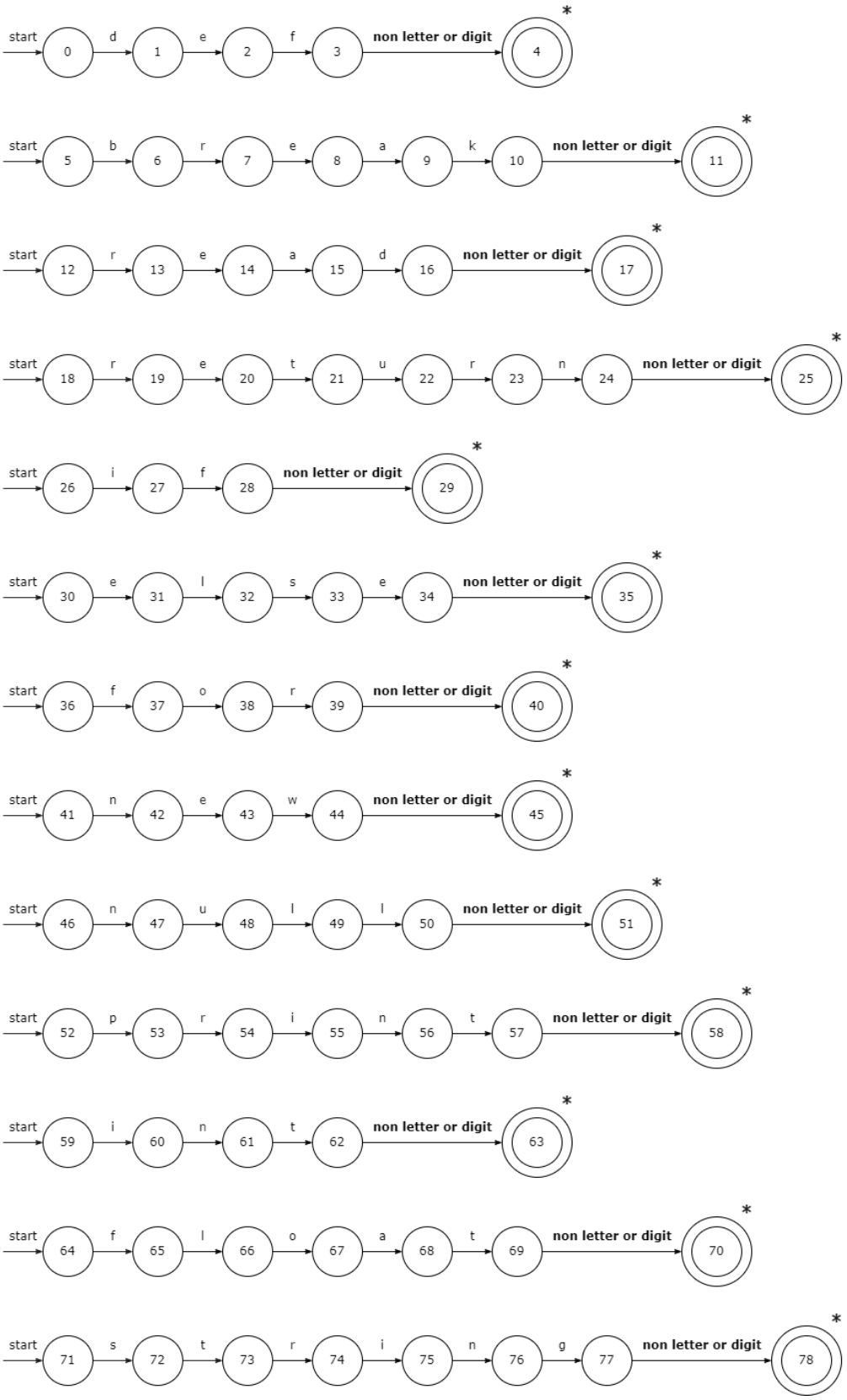
```
t_ignore = " \t"

def t_COMMENT(self, t):
       r"\/\/.*"
       pass
```

3 Diagramas de transição

3.1 Diagramas de transição para palavras reservadas

Na imagem abaixo estão representados os diagramas de transição para os tokens DEF, BREAK, READ, RETURN, IF, ELSE, FOR, NEW, NULL, PRINT, INT, FLOAT, e STRING respectivamente.

O termo *non letter or digit* é usado em todos os diagramas para representar qualquer caractere que não é uma letra, dígito numérico ou o caractere *underline* (“_”). Tal transição é necessária para garantir que as palavras reservadas não sejam identificadores, já que os identificadores são compostos apenas por letras, dígitos, ou caracteres underline.

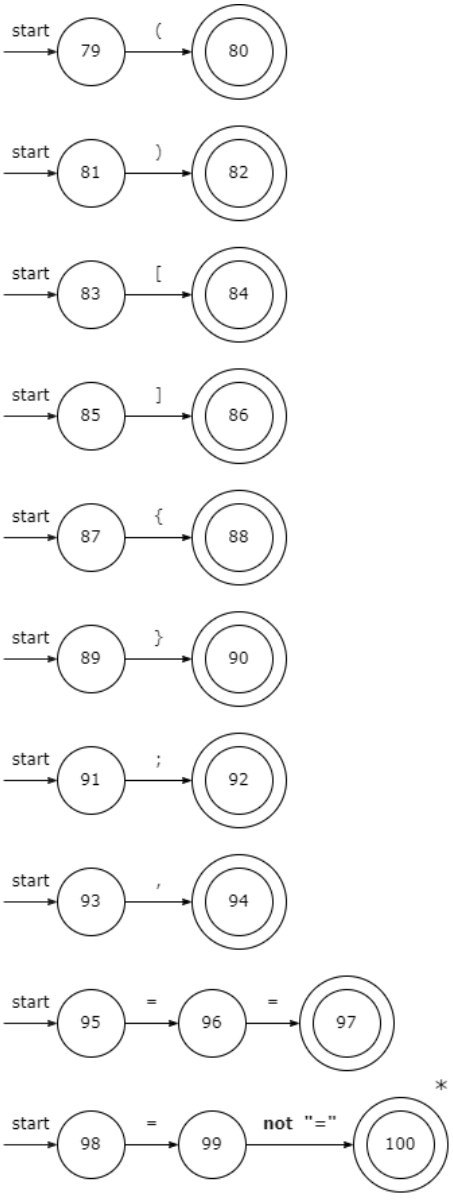

start
0
d
1
e
2
f
3
non letter or digit
4
*
start
5
b
6
r
7
e
8
a
9
k
10
non letter or digit
11
*
start
12
r
13
e
14
a
15
d
16
non letter or digit
17
*
start
18
r
19
e
20
t
21
u
22
r
23
n
24
non letter or digit
25
*
start
26
i
27
f
28
non letter or digit
29
*
start
30
e
31
l
32
s
33
e
34
non letter or digit
35
*
start
36
f
37
o
38
r
39
non letter or digit
40
*
start
41
n
42
e
43
w
44
non letter or digit
45
*
start
46
n
47
u
48
l
49
l
50
non letter or digit
51
*
start
52
p
53
r
54
i
55
n
56
t
57
non letter or digit
58
*
start
59
i
60
n
61
t
62
non letter or digit
63
*
start
64
f
65
l
66
o
67
a
68
t
69
non letter or digit
70
*
start
71
s
72
t
73
r
74
i
75
n
76
g
77
non letter or digit
78
*

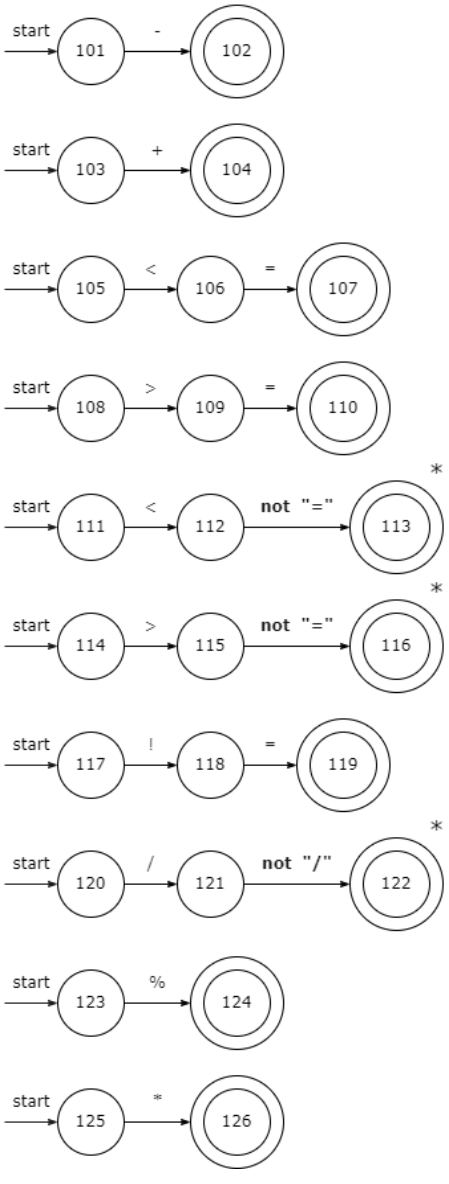
3.2 Diagramas de transição para símbolos terminais triviais

Nas imagens abaixo estão representados os diagramas de transição para os tokens LPARENTESES, RPARENTESES, LBRACKET, RBRACKET, LCURLYBRACKET, RCURLYBRACKET, SEMICOLON, COMMA, ASSIGNMENT, EQUAL, MINUS, PLUS, LESSTHAN, GREATERTHAN, LESSEQUAL, GRATEREQUAL, DIFFERENT, DIV, MOD e MULT respectivamente (lidos da esquerda para a direita e de cima para baixo). Foram criados termos para definir as sequências de caracteres relevantes de forma mais sucinta.

O termo *not "="* usado no diagrama para o token ASSIGNMENT, LESSTHAN e GREATERTHAN representa todo caractere que não é o símbolo de igualdade. Tal transição é necessária para diferenciar os três tokens de outros tokens similares.

O termo *not "/"* usado no diagrama para o token DIV representa todo caractere que não é o caractere para barra. Tal transição é necessária para diferenciar o token de divisão de um token de comentário.





3.3 Diagramas de transição para símbolos terminais não-triviais

Na imagem abaixo estão representados os diagramas de transição para os tokens IDENT, INT_CONSTANT, FLOAT_CONSTANT e STRING_CONSTANT respectivamente. Foram criados termos para definir as sequências de caracteres relevantes de forma mais sucinta. Além disso, no final dos diagramas é representada a ação que deve ser feita pelo analisador léxico para que seja possível inserir o valor dos tokens na tabela de símbolos.

O termo *letter* (letra) é usado para representar todos os caracteres que são considerados letras pelo alfabeto, maiúsculas ou minúsculas. Letter é equivalente à expressão regular [A-za-z].

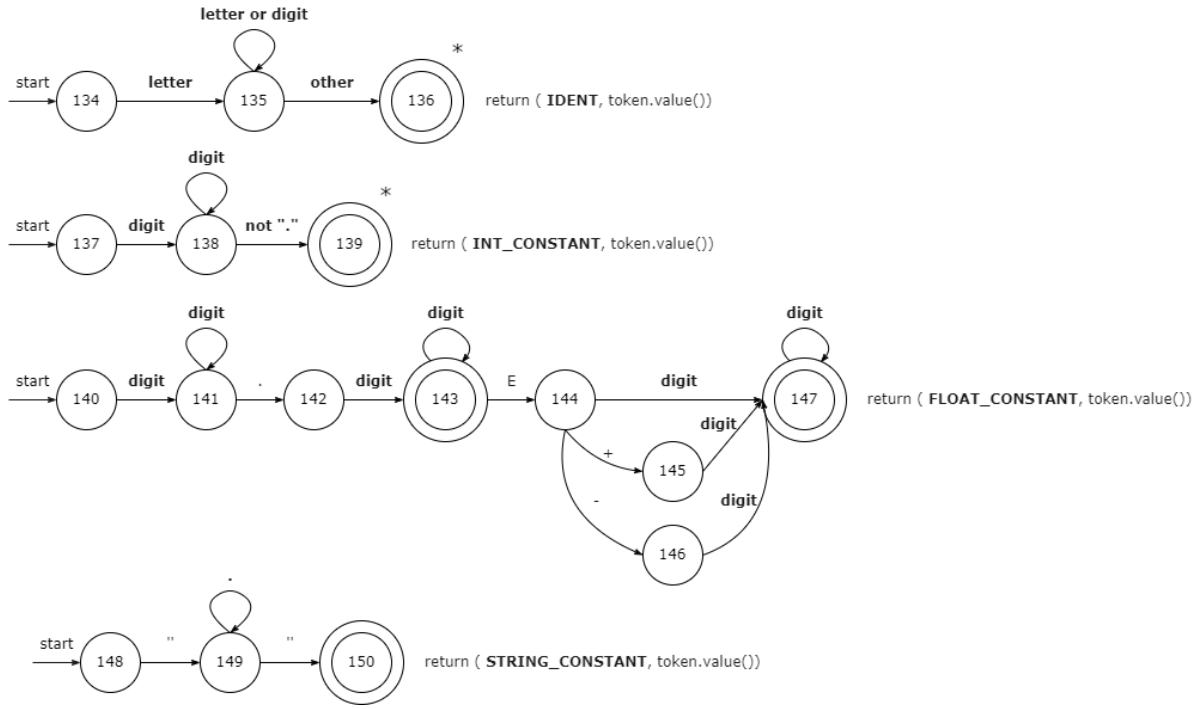
O termo *digit* (dígito) é usado para representar todos os dígitos numéricos. Ele é equivalente à expressão regular [0-9].

O termo *letter or digit* (letra ou dígito) representa todos os caracteres que são letras ou dígitos, conforme as definições acima, e também o caractere *underline* (“_”).

O termo *other* (outro) usado no diagrama para o token IDENT representa todos os caracteres que não são letras ou dígitos e, portanto, não podem ser continuações para um identificador, conforme a definição acima.

O termo *not “.”* usado no diagrama para o token INT_CONSTANT representa todo caractere que não é um ponto. Tal transição é necessária para garantir que o número é um inteiro e não um número em ponto flutuante.

O termo *“.”* usado no diagrama para o token STRING_CONSTANT representa qualquer caractere possível, da mesma forma como é usado na notação estendida de expressões regulares”.



3.4 Diagramas de transição para caracteres removíveis

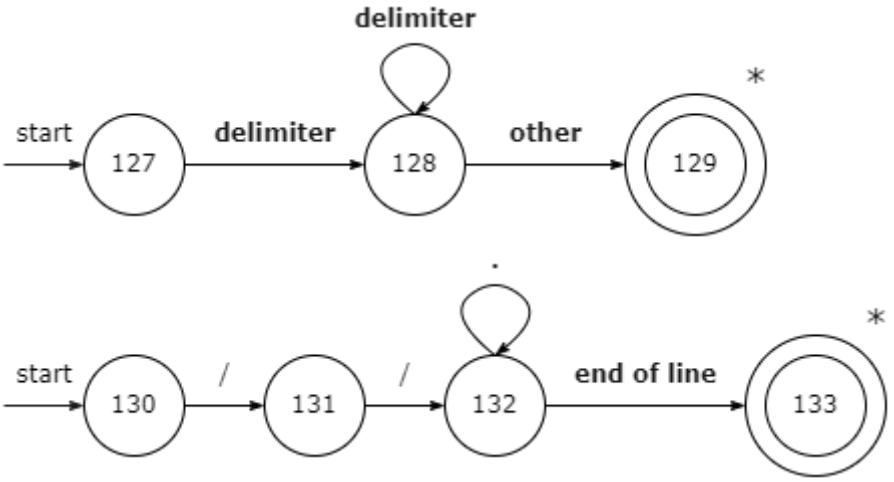
Na imagem abaixo estão representados os diagramas de transição para os tokens IGNORE e COMMENT respectivamente. Foram criados três termos para definir as sequências de caracteres relevantes de forma mais sucinta.

O termo *delimiter* (delimitador) é usado para representar todos os caracteres ou sequências de caracteres que fazem a separação entre os lexemas. No caso deste analisador léxico, tais caracteres são os especificados na expressão regular que define o token IGNORE, no caso, são o espaço em branco (“ ”) e a sequência “\t”, que representa a indentação feita usando Tab.

O termo *other* (outro) no diagrama para o token IGNORE representa todos os caracteres que não foram definidos como sendo delimitadores conforme a expressão regular que define o token IGNORE.

O termo “*.*” usado no diagrama para o token COMMENT representa qualquer caractere possível, da mesma forma como é usado na notação estendida de expressões regulares.

O termo *end of line* (fim de linha) é usado para representar a sequência de caracteres que define uma quebra de linha no arquivo de entrada para o analisador léxico. Por padrão para o PLY, tal sequência é “\n”.



4 Tabela de símbolos

A tabela de símbolos não é dada por padrão na ferramenta utilizada. Para obter da análise léxica os dados que consideramos relevantes ao processo de compilação, foi-se implementada uma estrutura de dados própria. Até o momento, no trabalho, somente armazenamos os identificadores encontrados na análise léxica.

4.1 Implementação

A implementação da tabela se deu por meio de um dicionário que continha em cada entrada a seguinte *dataclass* como entrada:

```
@dataclass
class SymbolTableEntry:
	var_name: str
	token_id: int
	line_declared: int
	lines_referenced: List[int]
	var_type: str = "Unknown"
```

4.2 Armazenamento dos símbolos

A princípio, esta estrutura de dados contém somente informações triviais obtidas a partir da análise léxica. Veja que a *dataclass* `SymbolTableEntry` armazena o valor do token (`var_name`), seu id na lista de tokens (`token_id`), a linha em que o token aparece pela primeira vez (`line_declared`), uma lista com as demais referências ao token (`lines_referenced`) e o tipo da variável que será armazenada nesta entrada, até o momento desconhecido.

Estas informações foram escolhidas porque, neste ponto do processo de compilação, dificilmente temos mais informações acerca dos identificadores.

5 Descrição da entrada e saída

O grupo decidiu utilizar a ferramenta *Python Lex-Yacc (PLY)* para a elaboração do trabalho. Este módulo é, essencialmente, uma re-implementação das ferramentas *lex* e *yacc* em python, originalmente para o ensino de compiladores.

5.1 Entrada para a ferramenta

Para que o PLY gere o analisador léxico, deve-se inserir as regras da linguagem no próprio código em python. Tome o exemplo dado na própria documentação do PLY:

Primeiramente definimos nomes/tipos que cada token poderá levar, como uma n-tupla:

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

Depois, temos que definir as expressões regulares que identificam cada token que queremos encontrar, assim como eventuais regras ao encontrar uma nova linha do código-fonte ou erros, e caracteres *whitespace* que devem ser ignorados.

```
# 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)
```

Depois de implementar estas definições, devemos construir o analisador léxico e utilizar o método `token()` para encontrar o próximo token.

5.2 Saída da ferramenta

A saída se dá à medida que precisamos do próximo token, e depende da implementação do usuário, mas essencialmente, o que se tem é uma lista de todos os tokens encontrados, com seu identificador definido em `tokens`, a linha em que ele foi encontrado e a posição do início do token no array de caracteres dado como entrada (código fonte).

Um exemplo de como buscar todos os tokens identificados é dado neste *snippet*:

```
lexer = lex.lex()
data = "A + 15 - (B * pi)"
lexer.input(data)
while True:
       tok = lexer.token()
       if not tok:
              break
       print(tok)
```

De acordo com as regras estabelecidas na seção anterior, geramos o analisador léxico **lex**. Os tokens encontrados na análise léxica de `"A + 15 -  (B * pi)"` foram:

```
LexToken(NAME,'A',1,0)
```

```
LexToken(PLUS,'+',1,2)
LexToken(NUMBER,15,1,4)
LexToken(MINUS,'-',1,7)
LexToken(LPAREN,'(',1,9)
LexToken(NAME,'B',1,10)
LexToken(TIMES,'*',1,12)
LexToken(NAME,'pi',1,14)
LexToken(RPAREN,')',1,16)
```