Analise Lexica

Clodoaldo A. Basaglia da Fonseca[1]

[1]Universidade Tecnológica Federal do Paraná - Campus de Campo Mourão (UTFPR)
R. Rosalina Maria Ferreira, 1233–Campo Mourão –PR – Brasil

[2]Departamento de Ciência da Computação
Paraná, BR.

[3]Compiladores - BCC36B

clodoaldofonseca92@gmail.com

Abstract. *This paper will describe how the lexical analises was made to conforme the TPP hipotetical language describe in the Compilers discipline. Here its shown the use of PLY(Python Lex-Yacc) with the Python language, where we describe its tokens, reserved words and regular expressions that was used to analyse and identify the use of integers, doubles and cientific expressions on the code, aswell words as identifiers and such.*

Resumo. *Este artigo descreverá como as análises lexicais foram feitas de acordo com a linguagem hipotética TPP descrita na disciplina Compiladores. Aqui é mostrado o uso de PLY (Python Lex-Yacc) com a linguagem Python, onde descrevemos seus tokens, palavras reservadas e expressões regulares que foram usadas para analisar e identificar o uso de números inteiros, duplos e expressões científicas no código, além de palavras como identificadores e tal.*

1. Sobre o PLY

Ply nada mais é do que uma implementação do Lex e YACC para Python. Em suma, PLY usa a LR(Left to Right) para a analise da gramatica, provendo a maioria da caracteristicas do lex/yacc comum, com suporte de produções vazias, regras de precedencia, recuperação de erros e grámaticas ambiguas. Podendo ser utilizado para uma extença checagem de erros.

2. O Ambiente

Para esse artigo, o ambiente utilizado para desenvolvimento foi contruido em um computador das seguintes configurações:

- Intel Core I5 7400 Coffe Lake 3.0Ghz
- 16Gb de memória RAM DD4 2400mhz
- Windows 10 Pro 64bits

No quesito de software, foram utilizados:

- Python 3.8.2
- Pip 20.0.2
- PyCharm Community Edition
- Windows Powershell

O pacote do Ply necessita ser instalado, já que não é um pacote padrão do Python, ele pode ser baixado através desse link[1] ou instalado utilizando-se do Pip com o comando:

[1]https://www.dabeaz.com/ply/ply-3.11.tar.gz

```
PS pip install --user ply
```

Após conclusão, passamos a desenvolver a lexica.

3. A linguagem T++(Tpp

A linguagem desenvolvida expecificamente para a disciplina de programção chamada T++(T plus plus) é fortemente tipada, relativamente parecida com o portugol. Utiliza-se de ”palavras”para inicio e fim de blocos. Tem estruturas de repetição, condição, vetores, vetores bidimensionais e dois tipos: flutuante e inteiro, bem como um terceiro tipo subentendido, o void. Suporta a existencia de váriavies locais e globais dos tipos citados acima.

O token ':=' represeta atribuição, assim como por exemplo, na linguaguem Pascal. Igualdade é feita apenas com o token '=' ao invés dos '==' comuns em outras linguagens. Uma variavel é criada com seu tipo, seguido de dois pontos e o seu nome, e em caso de vetores, do seu tamanho:

flutuante: a

Como exemplo, podemos ver a implementação de um programa que utiliza a linguagem T++ para implementar a busca binária:

```
inteiro: vetor[20]

inteiro busca_binaria(inteiro: numero, inteiro: inicio, inteiro: fim)
        inteiro: i
        i:= 20
        se (vetor[i]==numero) entao
                retorna i
        fim
        se (inicio==fim) entao
                {n o encontrou}
        sen o
                se(vetor[i] < numero) ent o
                        busca_binaria(numero,inicio+1,fim)
                sen o
                        busca_binaria(numero,inicio,i-1)
                fim
        fim
fim


inteiro principal()
        inteiro: i
        i:=0
        repita
                vetor[i] := i
                i := i + 1
```

```
        at   i = 20
        leia(numero)
        escreva(busca_binaria(numero,0,20))
        retorna 0
fim
```

4. A implementação

Inicialmente, para tal, é necessário a importação do lex do Play, através da instrução:

```
import ply.lex as lex
```

Como todas as linguagens, a T++ também possiu termos restritos a linguagens que não devem ser utilizados pelo programador, a lista pode ser vista a seguir, que também incorpora os tokens:

```
    reservadas = {
      'inteiro': 'INTEIRO',
      'flutuante': 'FLUTUANTE',
      'retorna': 'RETORNA',
      'se':'SE',
      'sen o':'SENAO',
      'ent o':'ENTAO',
      'fim':'FIM',
      'at ':'ATE',
      'repita':'REPITA',
      'principal':'PRINCIPAL',
      'leia':'LEIA',
      'escreva':'ESCREVA'
}
tokens = [
                'SOMA', 'SUBTRACAO', 'MULTIPLICACAO',
                'DIVISAO', 'IGUALDADE',
                'MAIOR', 'MENOR', 'MAIOR_IGUAL',
                'MENOR_IGUAL', 'ABRE_PAR', 'FECHA_PAR',
                'ABRE_COL', 'FECHA_COL', 'IDENTIFICADOR',
                'NEGACAO', 'DOIS_PONTOS', 'ATRIBUICAO',
                'VIRGULA','ABRE_CHAVES','FECHA_CHAVES',
                'COMENTARIO','NOTACAO_CIENTIFICA'
            ] + list(reservadas.values())
```

Além disso, é necessário declarar os tokens responsaveis pelos simbolos utilziados no código, como colchetes, parenteses, maior, maior igual, menor, menor igual e por assim em diante, como pode ser visto na lista abaixo:

```
t_ABRE_CHAVES=r'\{'
t_FECHA_CHAVES = r'\}'
t_SOMA = r'\+'
t_SUBTRACAO = '-'
```

```
t_MULTIPLICACAO = '\*'
t_DIVISAO = r'\/'
t_IGUALDADE = r'\='
t_MAIOR = r'\>'
t_MENOR = r'\<'
t_MAIOR_IGUAL = r'>='
t_MENOR_IGUAL = r'<='
t_ABRE_PAR = r'\('
t_FECHA_PAR = r'\)'
t_ABRE_COL = r'\['
t_FECHA_COL = r'\]'
t_NEGACAO = r'!'
t_DOIS_PONTOS = r':'
t_ATRIBUICAO = r':\='
t_VIRGULA = r'\,'
t_ignore = ' \t'
```

Esses tokens serão responsaveis por reconhecer os simbolos, variavies e os demais tokens do código, para alguns mais complexos, são necessárias expressões regulares, tais quais:

```
r'[a-zA-Z _][0-9_a-z _A-Z]*'
```

Responsavel por reconhecimento dos Identificadores, que nada mais são que nomes de variavies e nomes de métodos. Essa expressão aceita palavras iniciadas com letras, seguidas de numeros e/ou outras letras, sendo essas maiusculas ou não, com ou sem acento.

```
r'[+|-]?\d+\.\d+'
```

Já essa, responsavel por reconhecer os tipos "flutuantes", onde pode ter um simbulo positovo ou negativo, seguidos por digitos, um ponto, e outros digitos.

```
r'[+|-]?\d+'
```

Essa expressão acima, responsavel por reconhecer números tantos positivos quanto negativos, de forma inteira.

```
r'[+|-]?[0-9]+(\.[0-9]+)(e(\+|\-)?(\d+))?'
```

Reconhecendo notações cientificas através da expressão acima, no modelo exemplificado: +1.2e-9

```
r'\{.*?[^\}]+\}'
```

Para reconhecer comentários em uma linha, ou de várias.

Por fim, monta-se a estrutura do código seguindo as listas de tokens e palavras reservadas, bem como as expressões regulares e construindo o lexer junto com Python.

```
def __init__(self):
        self.lexer = lex.lex(debug=False, module=self)
```

Criando o lex com o debug desligado.

5. Resultados

Após a construção do código, podemos passar um arquivo escrito em T++ para o lex afim de extrair seus tokens e verificar se existem algum tipo de caracter não reconhecido.

```
PS python lexico.py expressao.tpp
```

Neste arquivos colocaremos as seguintes expressões afim de exemplificar a saída:

```
teste := (5+10)*14
    teste := 5+10*14
```

Com a seguinte saída:

```
PS C:\Users\Clodoaldo Basaglia\Documents\UTF2020\bcc36bic6a> python
Lexicanum
LexToken(IDENTIFICADOR,'teste',1,0)
LexToken(ATRIBUICAO,':=',1,6)
LexToken(ABRE_PAR,'(',1,9)
LexToken(INTEIRO,'5',1,10)
LexToken(INTEIRO,'+10',1,11)
LexToken(FECHA_PAR,')',1,14)
LexToken(MULTIPLICACAO,'*',1,15)
LexToken(INTEIRO,'14',1,16)
LexToken(IDENTIFICADOR,'teste',2,19)
LexToken(ATRIBUICAO,':=',2,25)
LexToken(INTEIRO,'5',2,28)
LexToken(INTEIRO,'+10',2,29)
LexToken(MULTIPLICACAO,'*',2,32)
LexToken(INTEIRO,'14',2,33)
```

6. Conclusão

Por fim, concluímos que é possivel criar um analisador léxico para a linguagem T++ utilizando Python 3.8 e Ply e extrair os tokens de um algoritmo escrito com a mesma.