Projeto de Implementação de um Compilador para a Linguagem T++

Análise Léxica

Alan Rodrigo Patriarca Santana[1]

[1]Departamento de Computação -- Universidade Tecnológica Federal do Paraná(UTFPR)

Caixa Postal 271 -- 87301-899 -- Campo Mourão -- PR -- Brazil

alan@alunos.utfpr.edu.br

Abstract. *This article tests the lexical functioning of a compiler developed in Python, which was built to perform the lexical analysis process of the T ++ language. The first compilation step that is covered in this article, refers to the process of reading the source code, with an input character file and a classification in a set of tags, tokens named as output. In this process we use a tool for Python compilers, Ply, in addition to also using resources such as regular expressions to obtain the tokens.*

Resumo. *Este artigo descreve o funcionamento léxico de um compilador desenvolvido em linguagem Python, que foi construído para realizar o processo de análise léxica da linguagem T++. O primeiro passo de compilação que é tratado nesse artigo, refere-se ao processo de leitura do código fonte, com um arquivo de caracteres de entrada e a classificação em um conjunto de marcas, denominados tokens como saída. Neste processo utilizamos uma ferramenta para compiladores em Python, o Ply, além de também recorrer a recursos como as expressões regulares para se obter os tokens.*

1. Introdução

Na ciência da computação, a análise léxica é basicamente o processo de se converter uma sequência de caracteres em um sequência de tokens. O processo funciona de forma a se ler os caracteres dispostos em um arquivo de entrada, fazer a sua classificação em "classes" e retornar a saída com os tokens classificados, ou seja, o processo é constituído basicamente de duas partes, a leitura dos caracteres e por fim a classificação em tokens. O processo será melhor descrito com exemplos a seguir.

2. Analizador Léxico

Nesta seção, será descrito em tópicos o funcionamento do analisador léxico, com exemplo de entrada, saída e os tokens utilizados para fazer a classificação dos caracteres e também uma breve descrição a respeito das expressões regulares.

2.1 Especificação dos tokens da Linguagem T++

Além das palavras reservadas e dos símbolos que serão mostrados em tabelas a seguir, existem mais alguns tipos de tokens, como os numéricos, identificadores e também os comentários.

Palavras Reservadas
se
então
senão
fim
repita
flutuante
retorna
até
leia
escreva
inteiro

Tabela 1. Tokens referentes as palavras reservadas

Símbolo	Identificação
+	soma
-	subtração
*	multiplicação
/	divisão
=	igualdade
,	vírgula
:=	atribuição

<	menor
>	maior
<=	menor igual
=>	maior igual
(	abre parentese
)	fecha parentese
:	dois pontos
&&	e lógico
| |	ou lógico
!	negação
[	abre colchete
]	fecha colchete

Tabela 2.Tokens referentes aos símbolos

Número: um ou mais dígitos, podendo ser inteiros ou em ponto flutuante, no qual a representação pode ser em notação científica ou não.

Identificador: começa com uma letra e precede com N letras e números. Este não possui limite de tamanho.

Comentários: os comentários são caracterizados por chaves, {...}.

2.2. Implementação da varredura e Ferramentas utilizadas

Este tópico trata de como foi o processo de desenvolvimento do analisador léxico, comentando e descrevendo o seu funcionamento, além também de relatar sobre a biblioteca e ferramentas utilizadas, que nesse caso foi o Ply.

2.2.1 Ferramentas e Bibliotecas

A biblioteca utilizada para o desenvolvimento do analisador léxico foi o Ply, este que é totalmente implementado em Python e traz várias ferramentas para se trabalhar com análise lex e também yacc.

A biblioteca citada trabalha com um outro conceito importante para fazer as varreduras, expressões regulares, sendo caracterizado por ser uma forma de identificar caracteres, como palavras reservadas ou até mesmo símbolos da linguagem.

2.2.2. Desenvolvimento

O desenvolvimento se iniciou basicamente com uma leitura aprofundada da documentação do Ply, para conseguir entender melhor os parâmetros a serem passados e também o seu funcionamento. Após essa análise inicial foi dado inicio no desenvolvimento prático do código do projeto.

Inicialmente foi realizado a leitura de um determinado arquivo contendo o código da linguagem t++. Em seguida foi criada a classe MyLexer, para o qual passaremos o código do arquivo como parâmetro para conseguirmos obter os tokens como saída.

Com a classe MyLexer, foi criado uma lista, contendo os principais tokens, além também de um dicionário com as palavras reservadas da linguagem.

As funções para verificação do token devem ser criadas com o mesmo nome do token, mas com um **t_** no início. É dessa forma que o Ply faz a relação dos tokens. Essas funções, basicamente são as expressões regulares para identificar cada um dos tokens.

Token	**Expressão regular**
MAIS	r'\+'
MENOS	r'-'
MULTIPLICACAO	r'*'
DIVISÃO	r'/ '
DOIS_PONTOS	r':'
VIRGULA	r','
MENOR	r'<'
MAIOR	r'>'
IGUAL	r'='
DIFERENTE	r'<>'
MENOR_IGUAL	r'<='
MAIOR_IGUAL	r'>='

E_LOGICO	r'&&'
OU_LOGICO	r'||'
NEGACAO	r'!'
ABRE_PARENTESE	r'\('
FECHA_PARENTESE	r'\)'
ABRE_COLCHETE	r'\['
FECHA_COLCHETE	r'\]'
NUM_NOTACAO_CIENTIFICA	r'[-\+]?((\d+\.\d*)|(\d*\.\d+)|(\d+))e[-\+]?((\d+\.\d*)|(\d*\.\d+)|(\d+))'
NUM_PONTO_FLUTUANTE	r'[-\+]?(\d+\.\d*)|(\d*\.\d+)'
NUM_INTEIRO	r'[-\+]?[\d]+'
COMMENT	r'\{.*?\}'
ID	r'[a-zA-Z_][a-zA-Zà-úÀ-Ú_0-9]*'
PALAVRA_RESERVADA	r'senão | se | então | fim | repita | até | leia | escreva | retorna | inteiro | flutuante'

Então, basicamente o Ply tem como entrada o código fonte, percorre os caracteres aplicando as expressões regulares acima e quando encontra alguma correspondência, retorna o token. Caso seja encontrado algum caractere ilegal, também foi desenvolvida uma função para tratar os erros. \newline

Para finalizar tanto o valor quanto o tipo do token são repassados para uma função que realiza a exibição destes no console.

2.3. Exemplo do sistema de varredura

Está seção mostra um exemplo de como o sistema funciona na prática, com o arquivo de entrada contendo o código na linguagem T++, e também o resultado de saída após o processamento dos tokens, com os seus respectivos valores e tipos.

2.3.1. Exemplo de entrada

```
inteiro: 11
flutuante: a[10]

inteiro fatorial(inteiro: n)
    inteiro: fat
    se n > 0 então {não calcula se n > 0}
        fat := 1
        repita
            fat := fat * n
            n := n - 1
        até n = 0
        retorna(fat) {retorna o valor do fatorial de n}
    senão
        retorna(0)
    fim
fim

inteiro principal()
    leia(n)
    escreva(fatorial(n))
    retorna(0)
fim
```

2.3.2 Modo de execução

Para executar o projeto deve-se passar o caminho do arquivo contendo o código em T++ como argumento na execução, como no exemplo abaixo.

```
python3 lex.py Exemplos/fat.tpp
```

2.3.3. Exemplo de saída

```
[inteiro] : [INTEIRO]
[:] : [DOIS_PONTOS]
[11] : [NUM_INTEIRO]
[flutuante] : [FLUTUANTE]
[:] : [DOIS_PONTOS]
[a] : [ID]
[[] : [ABRE_COLCHETE]
[10] : [NUM_INTEIRO]
```

```
[]] : [FECHA_COLCHETE]
[inteiro] : [INTEIRO]
[fatorial] : [ID]
[(] : [ABRE_PARENTESE]
[inteiro] : [INTEIRO]
[:] : [DOIS_PONTOS]
[n] : [ID]
[)] : [FECHA_PARENTESE]
[inteiro] : [INTEIRO]
[:] : [DOIS_PONTOS]
[fat] : [ID]
[se] : [SE]
[n] : [ID]
[>] : [MAIOR]
[0] : [NUM_INTEIRO]
[então] : [ENTAO]
[fat] : [ID]
[:] : [DOIS_PONTOS]
[=] : [IGUAL]
[1] : [NUM_INTEIRO]
[repita] : [REPITA]
[fat] : [ID]
[:] : [DOIS_PONTOS]
[=] : [IGUAL]
[fat] : [ID]
[*] : [MULTIPLICACAO]
[n] : [ID]
[n] : [ID]
[:] : [DOIS_PONTOS]
[=] : [IGUAL]
[n] : [ID]
[-] : [MENOS]
[1] : [NUM_INTEIRO]
[até] : [ATE]
[n] : [ID]
[=] : [IGUAL]
[0] : [NUM_INTEIRO]
[retorna] : [RETORNA]
[(] : [ABRE_PARENTESE]
[fat] : [ID]
[)] : [FECHA_PARENTESE]
[senão] : [SENAO]
[retorna] : [RETORNA]
[(] : [ABRE_PARENTESE]
```

```
[0] : [NUM_INTEIRO]
[)] : [FECHA_PARENTESE]
[fim] : [FIM]
[fim] : [FIM]
[inteiro] : [INTEIRO]
[principal] : [ID]
[(] : [ABRE_PARENTESE]
[)] : [FECHA_PARENTESE]
[leia] : [LEIA]
[(] : [ABRE_PARENTESE]
[n] : [ID]
[)] : [FECHA_PARENTESE]
[escreva] : [ESCREVA]
[(] : [ABRE_PARENTESE]
[fatorial] : [ID]
[(] : [ABRE_PARENTESE]
[n] : [ID]
[)] : [FECHA_PARENTESE]
[)] : [FECHA_PARENTESE]
[retorna] : [RETORNA]
[(] : [ABRE_PARENTESE]
[0] : [NUM_INTEIRO]
[)] : [FECHA_PARENTESE]
[fim] : [FIM]
```

3. Conclusão

Com este projeto foi possível fazer o tratamento do código fornecido na linguagem T++ e a geração dos tokens com base na entrada. Posteriormente os tokens utilizados nessa fase do compilador serão repassados para a segunda fase denominada análise sintática.

4. References

Ply, https://www.dabeaz.com/ply/ . Online, accessado 08-10-2020.

Expressões regulares, https://pt.wikipedia.org/wiki/Express%C3%A3o_regular. Online, accessado 08-10-2020.