Relatório do Analisador Léxico

Brendow P. C. Isidoro[1]

[1]Dacom - Universidade Técnológica Federal do Paraná (UTFPR)
Campo Mourão – PR – Brasil

brendowisidoro@alunos.utfpr.edu.br

Resumo. *Este documento descreve a implementação de um analisador léxico para a disciplina de compiladores do curso de ciência da computação. O código tem como objetivo reconhecer arquivos de entrada da linguagem T++, classificando as palavras encontradas nos arquivos em "tokens" que possam ser utilizados para implementações futuras.*

1. Introdução

Podemos definir um compilador como um tradutor de código fonte em alto nível para o código de máquina em baixo nível. Uma grande parte das linguagens disponíveis para os desenvolvimento de softwares e aplicativos utilizam compiladores, facilitando o desenvolvimento de novas ferramentas por abstrair conceitos complexos em funções simplificadas determinadas na documentação de cada linguagem.

A questão então é, como interpretar essas linguagens de alto nível, para algo compreensível pro computador? A implementação de um compilador é divida em várias camadas que realizam uma varredura do código e interpretam o contexto dele, dividindo-se em quatro partes principais, a (*i*)análise léxica, (*ii*)sintática, (*iii*)semântica e a (*iv*)geração de código intermediário.

As seções apresentadas a seguir neste relatório, descrevem a implementação de um analisador léxico, realizando a classificação das palavras encontradas no código fonte dos programas em T++ em *"tokens"*.

2. T++

O T++ é uma linguagem que se assemelha ao C/C++ sendo totalmente em português e descrita especificamente para o desenvolvimento de uma atividade avaliativa. Ou seja a linguagem é simplificada, não havendo a implementação de *Strings* como em outras linguagens.

A linguagem é quase fortemente tipada, sendo necessário declarar os tipos das variáveis quando são inicializadas e os tipos que podemos definir são o "inteiro"e "flutuante", sendo capaz de reconhecer notações científica também. Dizemos que ela é quase fortemente tipada pois nem todos os erros sao especificados porém eles são informados.

A Figura 1 apresenta os símbolos e palavras reservadas aceitas na linguagem T++ e são definidas por *tokens* que serão utilizados posteriormente.

A Figura 2 mostra a implementação do cálculo de fatorial, implementado em T++.

Figura 1. Tabela de *tokens*

palavras reservadas	símbolos
`se`	+ soma
`então`	- subtração
`senão`	* multiplicação
`fim`	/ divisão
`repita`	= igualdade
`flutuante`	, vírgula
`retorna`	:= atribuição
`até`	< menor
`leia`	> maior
`escreva`	<= menor-igual
`inteiro`	>= maior-igual
	(abre-par
	(fecha-par
	: dois-pontos
	[abre-col
	] fecha-col
	&& e-logico
	|| ou-logico
	! negação

Figura 2. Exemplo de Fatorial em T++

```
1   inteiro: n
2   flutuante: a[10]
3
4   inteiro fatorial(inteiro: n)
5       inteiro: fat
6       se n > 0 então {não calcula se n > 0}
7           fat := 1
8           repita
9               fat := fat * n
10              n := n - 1
11          até n = 0
12          retorna(fat) {retorna o valor do fatorial de n}
13      senão
14          retorna(0)
15      fim
16  fim
17
18  inteiro principal()
19      leia(n)
20      escreva(fatorial(n))
21      retorna(0)
22  fim
```

3. Implementação

3.1. Expressões regulares

Para a implementação do analisador léxico, é necessário definirmos expressões regulares que definem os padrões aceitos para reconhecer nomes e números.

As gramáticas podem ser demonstradas por autômatos, assim as Figura 3, Figura 4, Figura 5, Figura 6 descrevem as regras para reconhecer nomes de variáveis e funções, números inteiros, flutuantes e notação científica.

Figura 3. Autômato para ID

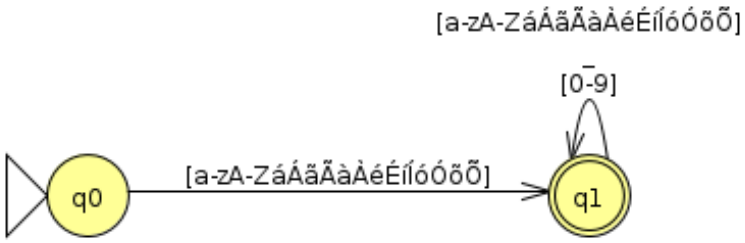


Figura 4. Autômato para Inteiro

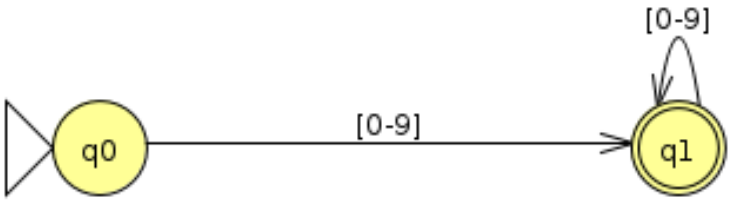
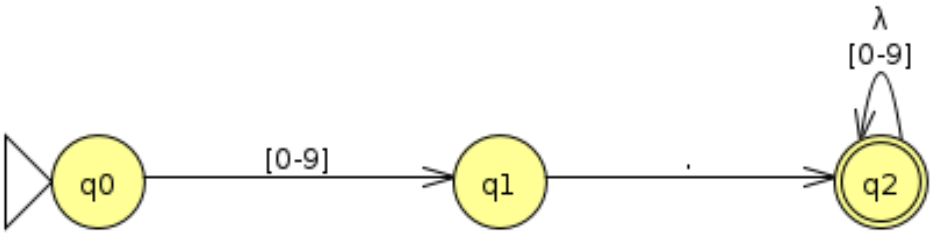


Figura 5. Autômato para Flutuante



As demais regras são apenas sequências de um ou dois caracteres, e palavras reservadas descritas na Figura 1.

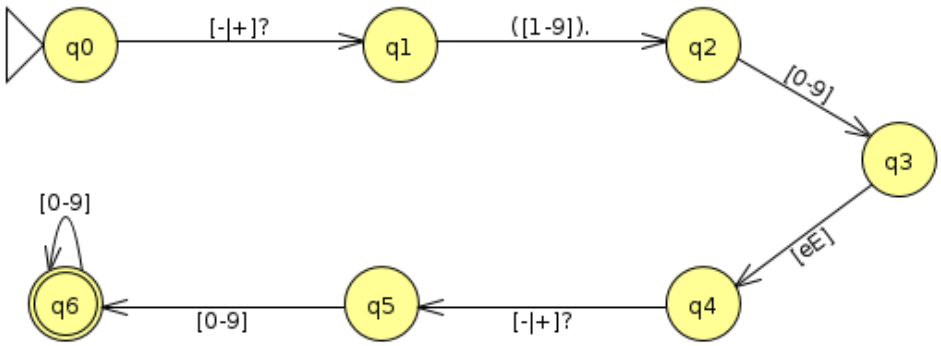
3.2. Ferramentas

A implementação do analisador léxico utiliza a linguagem de programação Python, na qual possui a biblioteca PLY que facilita a implementação dos *tokens* e das gramáticas do sistema.

3.2.1. PLY

O PLY é uma biblioteca em Python na qual re-implementa o analisador léxico (Lex) e o analisador sintático (Yacc) da linguagem C.

Figura 6. Autômato para Notação Científica



4. Resultados

O programa após executar a varredura de um arquivo de código fonte realiza a impressão dos tipos dos *tokens*, a Figura 7 mostra a saída da execução do código fatorial da Figura 2.

Também foi realizado um teste automatizado disposto pelo professor, no qual verificava a saída do programa com a saída esperada e assim gerando um registro dos códigos que foram percorridos corretamente. A Figura 8 mostra o resultado do teste automatizado. Os arquivos que foram encontrados erros possuem erros inseridos de maneira proposital, verificando se a linguagem suporta algum caractere que não lhe foi definido.

Figura 7. Saída de execução de código fatorial

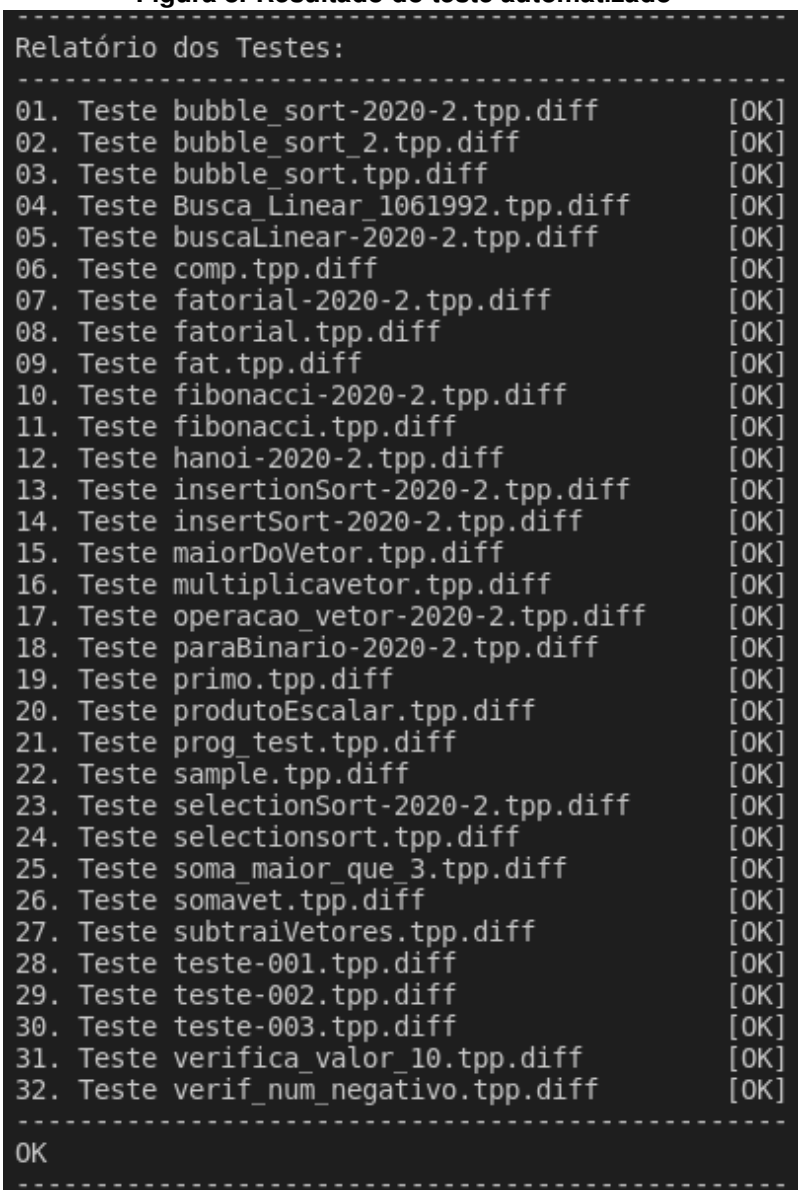
```
INTEIRO
DOIS_PONTOS
ID
FLUTUANTE
DOIS_PONTOS
ID
ABRE_COLCHETE
NUM_INTEIRO
FECHA_COLCHETE
INTEIRO
ID
ABRE_PARENTESE
INTEIRO
DOIS_PONTOS
ID
FECHA_PARENTESE
INTEIRO
DOIS_PONTOS
ID
SE
ID
MAIOR
NUM_INTEIRO
ENTAO
ID
ATRIBUICAO
NUM_INTEIRO
REPITA
ID
ATRIBUICAO
ID
MULTIPLICACAO
ID
ID
ATRIBUICAO
ID
MENOS
NUM_INTEIRO
ATE
ID
IGUAL
NUM_INTEIRO
RETORNA
ABRE_PARENTESE
ID
FECHA_PARENTESE
SENAO
RETORNA
ABRE_PARENTESE
NUM_INTEIRO
FECHA_PARENTESE
FIM
FIM
INTEIRO
ID
ABRE_PARENTESE
FECHA_PARENTESE
LEIA
ABRE_PARENTESE
ID
FECHA_PARENTESE
ESCREVA
ABRE_PARENTESE
ID
ABRE_PARENTESE
ID
FECHA_PARENTESE
FECHA_PARENTESE
RETORNA
ABRE_PARENTESE
NUM_INTEIRO
FECHA_PARENTESE
FIM
```

Referências

[Beazley] Beazley, D. M. Ply (python lex-yacc). http://www.dabeaz.com/ply/ply.html. Acessado: 2021-07-01.

[Gonçalves] Gonçalves, R. A. Bnf comentada. https://docs.google.com/document/d/1oYX-5ipzL_izj_hO8s7axuo2OyA279YEhnAItgXzXAQ/edit. Acessado: 2021-07-01.

Figura 8. Resultado do teste automatizado



```
-------------------------------------------------
Relatório dos Testes:
-------------------------------------------------
01. Teste bubble_sort-2020-2.tpp.diff        [OK]
02. Teste bubble_sort_2.tpp.diff             [OK]
03. Teste bubble_sort.tpp.diff               [OK]
04. Teste Busca_Linear_1061992.tpp.diff      [OK]
05. Teste buscaLinear-2020-2.tpp.diff        [OK]
06. Teste comp.tpp.diff                      [OK]
07. Teste fatorial-2020-2.tpp.diff           [OK]
08. Teste fatorial.tpp.diff                  [OK]
09. Teste fat.tpp.diff                       [OK]
10. Teste fibonacci-2020-2.tpp.diff          [OK]
11. Teste fibonacci.tpp.diff                 [OK]
12. Teste hanoi-2020-2.tpp.diff              [OK]
13. Teste insertionSort-2020-2.tpp.diff      [OK]
14. Teste insertSort-2020-2.tpp.diff         [OK]
15. Teste maiorDoVetor.tpp.diff              [OK]
16. Teste multiplicavetor.tpp.diff           [OK]
17. Teste operacao_vetor-2020-2.tpp.diff     [OK]
18. Teste paraBinario-2020-2.tpp.diff        [OK]
19. Teste primo.tpp.diff                     [OK]
20. Teste produtoEscalar.tpp.diff            [OK]
21. Teste prog_test.tpp.diff                 [OK]
22. Teste sample.tpp.diff                    [OK]
23. Teste selectionSort-2020-2.tpp.diff      [OK]
24. Teste selectionsort.tpp.diff             [OK]
25. Teste soma_maior_que_3.tpp.diff          [OK]
26. Teste somavet.tpp.diff                   [OK]
27. Teste subtraiVetores.tpp.diff            [OK]
28. Teste teste-001.tpp.diff                 [OK]
29. Teste teste-002.tpp.diff                 [OK]
30. Teste teste-003.tpp.diff                 [OK]
31. Teste verifica_valor_10.tpp.diff         [OK]
32. Teste verif_num_negativo.tpp.diff        [OK]
-------------------------------------------------
OK
-------------------------------------------------
```