Relatório Técnico do Problema 1: A Primeira Etapa de uma Tradução

Fabio Santos de Oliveira[1], **Joacy Mesquita da Silva**[1]

[1]Engenharia de Computação – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana, 09 de outubro de 2017

{fabiosantos1388,joacymsilva}@gmail.com

Abstract. *This report describes the Lexical Analyzer developed for the lexical structure of a previously established Language. The Analyzer was implemented in Java Language.*

Resumo. *Este relatório descreve o Analisador Léxico desenvolvido para a estrutura léxica de uma Linguagem estabelecida previamente. O Analisador foi implementado em Linguagem Java.*

1. Estrutura Léxica da Linguagem

A Estrutura Léxica da Linguagem pré-estabelecida está presente na Tabela 1.

Tabela 1. Estrutura Léxica da Linguagem

Palavras Reservadas	class, final, if, else, for, scan, print, int, float, bool, true, false, string
Identificador	letra { letra | dígito | _ }
Número	[-] Espaço dígito { dígito }[. dígito { dígito }]
Dígito	[0...9]
Letra	[a-z] | [A-Z]
Operadores Aritméticos	+, -, *, /, %
Operadores Relacionais	!=, =, <, <=, >, >=
Operadores Lógicos	!, &&, ||
Delimitador de Comentários	**// isso é um comentário de linha** **/*isso** **é um comentário de** **bloco*/**
Delimitadores	; , ( ) [ ] { } :
Cadeia de Caracteres	“{ Letra | Dígito | Símbolo | \” }”
Símbolo	ASCII de 32 a 126 (exceto ASCII 34)
Espaço	ASCII 9 | ASCII 10 | ASCII 13 | ASCII 32

1.1. Reconhecimento dos Tokens

De acordo com a estrutura léxica da linguagem recebida, foi possível explanar a forma de reconhecimento para classificação dos *tokens*. Essa forma de reconhecimento para cada *token* é informada nas subseções seguintes.

1.1.1. Palavra Reservada

As palavras reservadas são predefinidas, e seu reconhecimento é feito diretamente. Ao encontrarmos um *class*, *final*, *if*, *else*, *for*, *scan*, *print*, *int*, *float*, *bool*, *true*, *false* ou *string*, sabemos que se trata de uma palavra reservada.

1.1.2. Identificador

Um identificador pode ser uma letra, seja ela maíuscula ou minúscula. Também pode começar com uma letra, seguida de letras, dígitos ou *underlines* (_), não importando a quantidade deles.

1.1.3. Número

Um número pode ser negativo ou positivo, podendo iniciar ou não com '-', seguido de dígitos, caso seja um número inteiro, ou seguido de dígitos separados por '.', no caso de números decimais. No caso dos decimais deve haver pelo menos um dígito antes e depois do '.', no caso de inteiros deve haver pelo menos um dígito.

1.1.4. Operadores Aritméticos

Assim como as palavras reservadas, os operadores aritméticos são reconhecidos de forma direta. Ao encontrarmos '+', '-', '*', '/', ou '%' sabemos que se trata de um operador aritmético.

1.1.5. Operadores Relacionais

Os operadores relacionais também são reconhecidos de maneira direta. Ao nos depararmos com '!=', '=', '<', '<=', '>' ou '>=' podemos classificar como um operador relacional.

1.1.6. Operadores Lógicos

De maneira análoga a alguns citados acima, também os operadores lógicos são reconhecidos de modo direto. Quando encontramos '!', '&&' ou '||' sabemos se tratar de um operador lógico.

1.1.7. Delimitador de Comentários

Caso seja comentário de linha, deve iniciar com '//' e terminar com uma quebra de linha. Caso seja comentário de bloco, deve iniciar com '/*' e terminar com '*/', entre os dois delimitadores pode haver qualquer informação.

1.1.8. Delimitadores

De maneira direta podemos reconhecer os delimitadores. Caso encontremos ';', ',', '(', ')', '[', ']', '{', '}' ou ':' podemos classificar como um delimitador.

1.1.9. Cadeia de Caracteres

A cadeia de caracteres deve começar e terminar com aspas duplas, e entre essas aspas podem ter letras, números, símbolos ASCII de 32 a 126, exceto o ASCII 34, ou \".

1.1.10. Símbolo

De forma direta, todo ASCII de 32 a 126, exceto o ASCII 34 é reconhecido como símbolo.

1.1.11. Espaço

Os espaços reconhecidos são ASCII 9, ASCII 10, ASCII 13, ou ASCII 32.

2. Detecção de Erros Léxicos

2.1. Erro de Comentário de Bloco

No quesito de erros para comentários de bloco, ficou definido e reconhecido como erro comentários de bloco iniciados (/*) porém não sendo encontrado o seu término (*/). Por decisão de projeto foi definido também que os comentários de bloco não aceitará caracteres de barra(/) e asterisco(*) no seu interior, a não ser que apareçam juntos e nesta forma(/*), sendo esses caracteres válidos apenas para delimitarem o início e fim do bloco de comentário.

Ex.: Erro de comentário de bloco: "/* Comentário" . - Falta de fechamento de bloco; "/* Comentario / */" . - Caracter (/) no interior do comentário; "/* Comentario * / * */" . - Caracter (*) e/ou (/) no interior do comentário.

2.2. Erro de Identificador

No quesito de erros para identificador, foi considerado sequências de caracter começados por letras ([a-zA-Z]) seguidos de qualquer outro caracter que não fosse letra, dígito ([0-9]) ou *underline* (_). Assim como sequência de caracteres iniciados por símbolos ou números e seguidos de letras alternando-se ou não.

Ex.: Erro de identificador: "abc@"; "12aeR"; "@@abc2".

2.3. Erro de Número

No quesito de erros para números, foi considerado sequências de dígitos seguido de ponto (.) continuados por qualquer coisa diferente de dígito. Assim como sequências iniciadas por ponto (.) e seguidas de dígitos.

Ex.: Erro de numero: "154.@"; ".458".

2.4. Erro de Cadeia de Catateres

No quesito de erros de cadeia de caracteres, foi considerado abertura de aspas (") sem encontrar o fechamento, assim como não haver quebra de linha dentro da cadeia e aceitar aspas dentro da cadeia apenas precedido de barra invertida neste formato (\"). Tambem será considerado erro em casos de cadeiaa de caracter contendo simbolos que não pertençam aos simbolos válidos da linguagem.

Ex.: Erro de cadeia de Caracteres: "sdasdsad ; "ssjjsj"dskj"

2.5. Erro de Símbolo

No quesito erro de símbolo, foi considerado qualquer símbolo escrito fora do escopo de cadeia de caractere ou de comentários.

Ex.: Erro de simbolo: "@.@"; " . ".

3. Execução do Programa

Para a execução do analisador é necessario criar arquivos de entrada (ou utilizar os existentes) dentro do diretorio "entrada" situado no diretorio "Analisador_Léxico_Fabio_Joacy", a saída produzida pelo analisador será armazenada no diretorio "saida" também situado no diretório "Analisador_Léxico_Fabio_Joacy".

Para o reconhecimento do diretorio contendo os arquivos de entrada se faz necessario alterar a classe Executar.java, informando o caminho para o diretorio "entrada". Mais especificamente alterar as linhas 8 e 14 da classe Executar.java com um caminho semelhante ao exemplo a seguir:

Ex.: "C:\\Users\\fabio\\Documents\\eclipse-workspace\\Analisador_
Léxico_Fabio_Joacy\\entrada".

Em caso de utilizar arquivos de texto que contenham cê-cedilha (ç), é possível que ocorra um erro de leitura do arquivo.

4. Resultado da Análise

O resultado da análise é escrito em um arquivo de texto, onde cada linha representa um lexema, o tipo de *token* a qual pertence e a linha onde ele foi encontrado. O padrão de exibição é mostrado abaixo:
$<$ Token , Lexema $>$ Linha: Nº da linha

Ex.: Se for detectado um identificador ABC na linha 1, o resultado a ser exibido é:
$<$ Identificador, ABC $>$ Linha: 1