



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS CRATEÚS

DISCIPLINA: COMPILADORES

PROFESSORA: LISIEUX MARIE MARINHO DOS SANTOS ANDRADE

Projeto de Compiladores -Análise Léxica

Regras Gerais

O desenvolvimento do Analisador Léxico deve ocorrer com o uso da linguagem de programação Java, ficando o analisador responsável por criar a tabela de símbolos, onde cada entrada é formada pelos seguintes elementos:

- Token;
- Lexema;
- Linha correspondente a posição do token;
- Valor.

COMPONENTES E COMPORTAMENTOS

- I.** É vedado o uso de geradores automáticos nesta etapa;
- II.** Os seguintes elementos não devem ser considerados pelo analisador:
 - Espaços em branco;
 - Caracteres formatadores (tabulação, nova linha, novo parágrafo). Lembrar que alguns dos caracteres especiais servem para fazer a contagem da linha;
 - O analisador léxico também não deve considerar comentários.
- III.** Os seguintes tipos de token devem ser considerados:
 - Construção de escopo – Blocos de comandos (principal, condicional e de repetição);
 - Palavras-chaves – Comandos próprios da linguagem;
 - Identificadores: constantes e variáveis;
 - Operadores: lógicos e aritméticos;
 - Expressões: condicionais, laços de repetição, aritméticas e lógicas;
 - Delimitadores: ; (ponto e vírgula) . (ponto final) : (dois pontos) () (abre e fecha parênteses # (hashtag) @ (arroba);
 - Comando de Atribuição;
 - Tipos de dados: primitivos da linguagem (inteiros, reais, lógicos e vazio), strings e vetores.
- IV.** Os seguintes erros devem ser detectados pelo analisador léxico:
 - Comentário aberto e não fechado;
 - Símbolos não pertencentes a linguagem.

DAS ETAPAS

A construção do Analisador Léxico dará por meio de dois momentos

- I.** Desenvolver relatório (documento no padrão de artigo da SBC) que apresente a construção dos tokens da linguagem abordada por meio de expressões regulares e de diagrama(s) de transição(ões);
- II.** Implementar o analisador léxico com uso da linguagem Java, permitindo ao usuário ambiente para a inclusão do código fonte (importação ou campo editor), assim como visualização do processo de análise (retorno da detecção de erro).

SEÇÃO DE EXEMPLOS

Programa Fonte:

```
program teste; {programa exemplo}
begin
42 + (675 * 31) - 20925
end;
```

Tabela de Símbolos:

Token		Lexema	Linha
Tipo	Valor		
Palavra reservada	PROGRAM	program	1
Identificador	ID	teste	1
Pontuacao	VIR	;	1
Palavra reservada	BEGIN	begin	2
Numero	42		3
Operador	SOMA		3
Pontuacao	PARESQ		3
Numero	675		3
Operador	MULT		3
Numero	31		3
Pontuacao	PARDIR		3
Operador	SUB		3
Numero	20925		3
Palavra reservada	END	end	4
Pontuacao	VIR	;	4

Interface do Compilador:

The screenshot displays a compiler interface with three main sections:

- Source Code (Left):** A text editor showing the following code:

```
1] programa ;
2]
3] var i : integer;
4]   aux char;
5]
6] inicio
7]   escrevaln('teste');
8]   leialn();
9]
10] se aux <= teste
11]   entao leialn(aux)
12]   senao escrevaln('TEste');
13]
14] fim.
15]
```
- Token Stream (Right):** A table showing the sequence of tokens generated from the source code:

Token	Lexema	Linha
<PROGRAMA>	PROGRAMA	1
<PONTOVIRG>	;	1
<VAR>	VAR	3
Id1	i	3
<DOISPONTO>	:	3
Tipo1	INTEGER	3
<PONTOVIRG>	;	3
Id2	AUX	4
Tipo2	CHAR	4
<PONTOVIRG>	;	4
<INICIO>	INICIO	6
<ESCREVALN>	ESCREVALN	7
<ABREPARENTeses>	(7
Texto	'TESTE'	7
<FIM>	FIM	14
- Error Messages (Bottom):**
 - Erros Léxicos:** NENHUM_ERRO_ENCONTRADO
 - Erros Sintáticos:**
 - Esperando " IDENTIFICADOR " antes de ; na linha 1
 - Esperando " : " antes de <TIPO> na linha 4
 - Esperando " IDENTIFICADOR " antes de) na linha 8

Buttons at the bottom right: Analisar, Limpar Dados.