Departamento de Ciência da Computação - UFJF

DCC045 - Teoria dos compiladores

Analisador Léxico para a Linguagem *lang*

Alunos:

Daniel Souza Ferreira - 201665519B

Matheus de Oliveira Carvalho - 201665568C

Junho / 2021

1 Introdução

Neste trabalho desenvolveremos o analisador léxico para a criação de um compilador para a linguagem *'lang'*.

O analisador léxico, ou scanner, é a primeira fase do processo de compilação e tem como função transformar uma sequência de caracteres vindo de um arquivo de entrada em uma sequência de símbolos relevantes para o programa chamados de *Tokens*, descartando espaços em brancos e comentários no processo. *Tokens* é uma unidade sintática que contém lexema (uma expressão regular) e uma tag (uma informação utilizada para caracterizar o token) e que é enviada para o analisador sintático dando continuidade ao processo de compilação.

2 Características do Projeto

Para construção do analisador léxico utilizamos a ferramenta JFLEX, que consiste em um gerador de analisador léxico para Java escrito em Java. Utilizamos o formalismo de expressões regulares a fim de converter as mesmas em *Tokens* que serão indentificados se os caracteres lidos forem válidos conforme estruturado anteriomente. Os detalhes referentes a definição dos macros e a relação entre os lexemas e *Tokens*, e que servirá para construção do analisador léxico, esta presente no arquivo LexicalGenerator.jflex.

2.1 Expressões Regulares e Tokens

Para a utilização do JFLEX foram determinados 42 Tokens, que podem ser acessados através do arquivo TOKEN_TYPE.Java, referentes as palavras reservadas, operadores e separadores, literais inteiro, ponto flutuante, caractere e lógico. Vale ressaltar que para os literais inteiro, ponto flutuante e caractere foram definidos *Tokens* próprios (INTEGER, DOUBLE e CARACTER) e que também foram utilizadas algumas notações na tabela representando os *Tokens* abaixo.

- letter: Referente a letras entre A e Z
- digit: Referente a números entre 0 e 9

Nome do Token	Lexema
ID	letter + (letter | digit | _)*
INT	Int
FLOAT	Float
CHAR	Char
BOOL	Bool
TRUE	true
FALSE	false
NULL	null
LEFTPARENT	(
RIGHTPARENT	)
LEFTBRACE	[
RIGHTBRACE	]
LEFTBRACKET	{
RIGHTBRACKET	{
GREATER	>
LESS	<
DOT	.
COMMA	,
COLON	:
SEMICOLON	;
DOUBLECOLON	::
ASSIGN	=
EQ	==
NEQ	!=
PLUS	+
MINUS	-
MULT	*
DIV	/
MODULE	%
AND	&&
NOT	!
IF	if
ELSE	else
ITERATE	iterate
READ	read
PRINT	print
RETURN	return
NEW	new
INTEGER	digit+
DOUBLE	digit* + . + digit+
CARACTER	letter | \ | \\ | \n | \t | \b | \r

Tabela 1. Tabela de tokens e suas lexemas

3 Implementação do Trabalho

Como citado anteriormente o arquivo LexicalGenerator.jflex contém as expressões regulares que serão relacionadas aos *Tokens*, representadas pelos macros, sendo que os mesmos serão definidos no estado inicial do autômato. Foram definidos *states* para representar os comentários realizados em uma única linha, sendo iciados pela presença de "−−" e encerrados por uma quebra de linha, e em várias linhas, sendo iniciado por "{ −" e encerrado por "− }".

Em seguida será gerado um arquivo LexicalText.java que simulará o comportamento do AFD responsável por reconhecer as expressões regulares e relacionar as mesmas com os *Tokens* da linguagem. Uma classe principal Main em Java fará a leitura do arquivo enquanto uma classe auxiliar Token.Java irá receber os *Tokens* lidos e suas propriedades.

4 Testes Realizados

Para realizar os testes usamos os arquivos disponibilizados pelo professor. Dentre esses apresentaremos os resultados de attrADD e attrCHARESCAPE1, dos arquivos listados como errados, e o attrFALSE, dos listados como certos.

- attrADD: nesse exemplo, o código apresenta um erro sintático **(x = + 1 2)** porém como o analisador léxico não o reconhece, a geração de *Tokens* ocorre normalmente.

```
java Main errado/attrADD.lan
[(1,1) "ID" : <main>]
[(1,5) "(" : <>]
[(1,6) ")" : <>]
[(1,7) "{" : <>]
[(2,3) "ID" : <x>]
[(2,5) "=" : <>]
[(2,7) "+" : <>]
[(2,9) "INTEGER" : <1>]
[(2,11) "INTEGER" : <2>]
[(2,12) ";" : <>]
[(3,1) "}" : <>]
Total de tokens lidos 11
```

- attrCHARESCAPE1: o arquivo utilizado apresenta um erro do tipo léxico **(x =’\$ n’)** já que \$ não é um *Token* reconhecido portanto o analisador para sua execução, indicando o erro.

```
[(1,1) "ID" : <main>]
[(1,5) "(" : <>]
[(1,6) ")" : <>]
[(1,7) "{" : <>]
[(2,3) "ID" : <x>]
[(2,5) "=" : <>]
Exception in thread "main" java.lang.RuntimeException: Illegal character <'>
    at LexicalText.nextToken(LexicalText.java:971)
    at Main.main(Main.java:23)
makefile:8: recipe for target 'run' failed
make: *** [run] Error 1
```

- attrFALSE: esse arquivo de teste não possui nenhum tipo de erro, nem léxico nem sintático portanto a leitura ocorre normalmente.

```
java Main certo/attrFALSE.cmd
[(1,1) "ID" : <main>]
[(1,5) "(" : <>]
[(1,6) ")" : <>]
[(1,7) "{" : <>]
[(2,3) "ID" : <x>]
[(2,5) "=" : <>]
[(2,7) "false" : <>]
[(2,12) ";" : <>]
[(3,1) "}" : <>]
Total de tokens lidos 9
```

5 Execução do Programa

- Para gerar o .Java referente ao analisador léxico e compilar os arquivos necessários para o mesmo deve-se usar o comando: **make**
- Para executa-lo e fazer a leitura do arquivo *lan* deve-se usar o comando: **java Main *nome do arquivo que deseja ler***