Departamento de Ciência da Computação - UFJF
DCC045 - Teoria dos compiladores

Analisador Sintático para a Linguagem *lang*

Alunos:

Daniel Souza Ferreira - 201665519B

Matheus de Oliveira Carvalho - 201665568C

Julho / 2021

1 Introdução

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

O analisador sintático, ou parser, é a segunda fase do processo de compilação e tem como função verificar se a entrada é um problema válido ou não, utilizando na maioria dos casos gramáticas livres de contexto para especificar a sintaxe de uma linguagem de programação. Ela recebe uma sequência de tokens vindo do analisador léxico e determina se essa sequência pode ser gerada, emitindo um erro caso não possa.

2 Características do Projeto

Para a construção do analisador sintático foi utilizada a ferramenta ANTLR em sua versão 4.9.2, sendo a versão mais recente até a presente data. O ANTLR consiste em um gerador de analisadores sintáticos de linguagens, sendo possível utilizar o mesmo para tarefas como leitura, processamento, execução ou tradução de um texto que seja estruturado ou de arquivos em formato binário.

Devido ao fato de não ser possível realizar o interligamento entre o analisador léxico construído no trabalho anterior pela ferramenta JFLEX e o analisador sintático que será desenvolvido a partir do ANTLR, foi necessário a construção de outro analisador léxico utilizando a ferramenta ANTLR, reaproveitando os *Tokens* definidos anteriormente e possuindo alguns *Tokens* a mais do que o anterior (DATA e IDTYPE) a fim de completude do mesmo, podendo ser consultados na Tabela 1. Alguns procedimentos foram realizados de forma diferente, como a definição da expressão regular que será responsável por identificar comentários de linha e de bloco, com a ausência de criação dos *states* para tal como foi feito anteriormente.

2.1 Arquivos Presentes no Trabalho

- Node.java e SuperNode.java
- Main.java
- lang.g4
- Arquivos gerados pelo ANTLR
  - lang.interp
  - lang.tokens

  - langLexer.interp
  - langLexer.tokens
  - langLexer.java
  - langParser.java
  - langBaseListener.java
  - langListener.java
- ParseAdaptor.java
- TestParser.java
- LangCompiler.java

3 Implementação do Trabalho

Como citado anteriormente foi utilizado a ferramente ANTLR para a geração dos arquivos do analisador sintático e léxico passando os tokens e as regras da grámatica da lingua pelo arquivo lang.g4. Após os arquivos terem sido gerados foi necessário a criação da classe *Node.java* para instanciar a classe *SuperNode.java* fornecida pelo professor. Essa classe será útil na implementação da próxima etapa do compilador.

Para a execução do programa e seus testes foi necessário criar a classe *Main.java* que é uma implementação da interface *ParseAdaptor.java* fornecida pelo professor. Essa classe é responsável por transformar o arquivo de entrada em um parser e depois montar a ParserTree, que é setada em uma variável Node, caso não apresente erros sintáticos, sendo essa variavel passada para uma outra variavel criada na classe *LangCompiler.java*. A *LangCompiler.java*, que foi é fornecida pelo professor,é a classe chamada para a execução do programa, recebendo o valor da *Main.java* e chamando a classe de testes *TestParser.java* que ira realizar a báteria de testes.

Nome do Token	Lexema
ID \ IDTYPE	[a-z] [a-zA-Z0-9]* \ [A-Z] [a-zA-Z0-9]*
DATA	data
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	(0...9)+
DOUBLE	(0...9)* ’.’ (0...9)+
CARACTER	’\” ([\\’] | ’\\n’ | ’\\t’ | ’\\b’ | ’\\r’ | ’\\\\’ | ’\\\”) ’\”

Tabela 1. Tabela de tokens atualizadas e suas lexemas

4 Execução do Programa

Para a execução do programa foi criado um makefile que possui os seguintes comandos:

- **make** - utilizado para fazer a limpeza dos arquivos .class gerados na pasta 'lang' e depois realizar a compilação de todos os arquivos .java.
- **make clean** - utilizado para fazer a limpeza dos arquivos .class gerados na pasta 'lang'.
- **make execute** - utilizado para realizar a execução da batéria de testes.

A batéria de testes gerada pelo make execute é dos arquivos da pasta 'certa'. Para realizar os testes dos arquivos da pasta 'errado' deve-se alterar o valor da variável **okSrcs** na classe **TestParser.java** para **"testes/sintaxe/errado/"**.