REO 02 - REOBoy (Atividade 2)

Por Pedro Antônio de Souza.

1 INTRODUÇÃO

O Gerador de Analisadores Léxicos e Sintáticos (GALS) é uma ferramenta de Software Livre desenvolvida por Carlos Eduardo Gesser, sob orientação do Prof. Olinto José Varela Furtado, como trabalho de conclusão do curso de Bacharelado em Ciências da Computação da Universidade Federal de Santa Catarina.

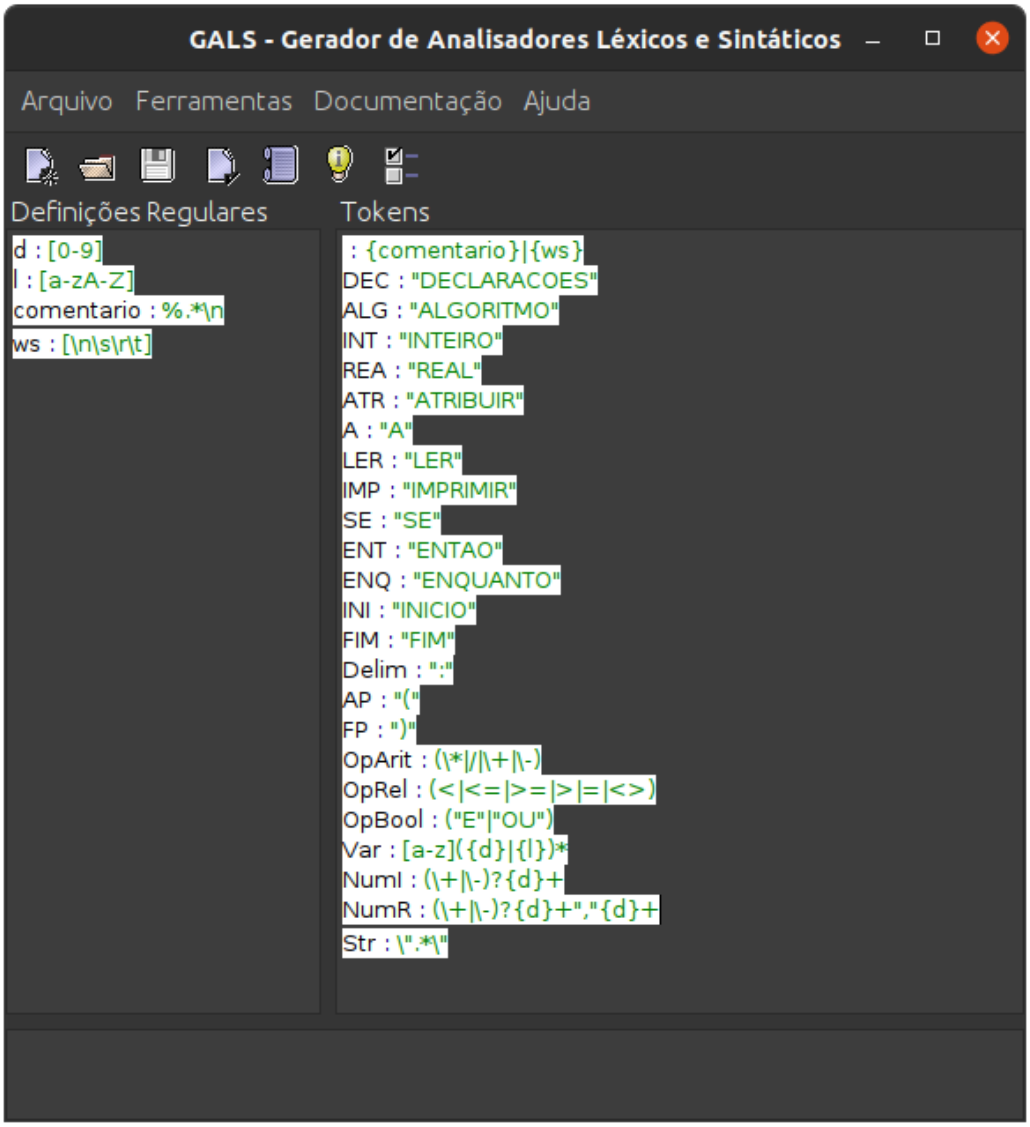
Nesse REO 02 – REOBoy, foi proposto a utilização do ambiente GALS para criar um analisador léxico (AL) para a linguagem ALGUMA. Assim, foi gerado o código do AL em Java e, posteriormente, foram executados testes utilizando os algoritmos de calculo de fatorial de N e soma de números da sequência de Fibonacci.

Esse relatório tem como objetivo descrever as expressões regulares (ERs) utilizadas no GALS, o código criado para o analisador e os resultados da execução, além de analisar e comparar os geradores de analisador léxico Lex e Flex.

2 EXPRESSÕES REGULARES

A ferramenta GALS permite especificar ERs como *Definições Regulares* (DRs) que possuirão um nome e uma ER atribuída. Essas definições são expressões que possuem uso constante no analisador e poderão ser utilizadas através de seu nome envolvido por chaves, isto é, uma definição regular `digito : [0-9]` poderá ser utilizada nas expressões utilizando `{digito}`.

A captura de tela a seguir exibe todas as ERs utilizadas na criação do analisador léxico.



As definições regulares utilizadas na geração do analisador léxico da linguagem ALGUMA estão ilustradas no quadro abaixo.

DR	DESCRIÇÃO
`d : [0-9]`	Define o conjunto dos dígitos de `0` a `9`.
`l : [a-zA-Z]`	Define o conjunto das letras de `a` a `z`, minúsculas e maiúsculas.
`comentario : %.*\n`	Define o padrão de comentário da linguagem: inicia-se com `%`, seguido de qualquer quantidade de qualquer caractere e termina com uma quebra de linha.
`ws : [\n\s\r\t]`	Define o conjunto de espaços em branco.

Entre os tokens `DEC` e `FP`, as ERs reconhecedoras são cadeias de caracteres literais, ou seja, serão reconhecidos apenas os textos específicos que foram definidos entre aspas. Nos demais, utilizou-se expressões regulares que são elucidadas no quadro a seguir.

TOKEN	DESCRIÇÃO DA ER
`: {comentario}|{ws}`	Serão descartados os tokens reconhecidos como comentário ou espaço em branco.
`OpArit : (\*|/|\+|\-)`	Define o conjunto de operadores aritméticos que são `*`, `/`, `+` ou `-`.
`OpRel : (<|<=|>=|>|=|<>)`	Define o conjunto de operadores relacionais que são `<`, `<=`, `>=`, `>`, `=` ou `<>`.
`OpBool : ("E"|"OU")`	Define o conjunto dos operadores booleanos `E` ou `OU`.
`Var : [a-z]({d}|{l})*`	Define o padrão de variáveis da linguagem ALGUMA: inicia-se com uma letra minúscula e segue-se com qualquer quantidade de dígitos ou letras.
`NumI : (\+|\-)?{d}+`	Define o padrão de números inteiros: pode possuir ou não um sinal positivo ou negativo, seguido de um ou mais dígitos.
`NumR : (\+|\-)?{d}+","{d}+`	Define o padrão de números reais: pode possuir ou não um sinal positivo ou negativo, seguido de um ou mais dígitos, seguido de uma vírgula e, por fim, segue-se um ou mais dígitos.
`Str : \".*\"`	Define o padrão de *string*: qualquer quantidade de qualquer carácter entre aspas.

3 CÓDIGO GERADO PARA O ANALISADOR

Após a definição dos tokens da linguagem, o GALS gera uma série de classes em Java (opção escolhida para a atividade), sendo eles:

1. `AnalysisError` – Classe que herda de `Exception` utilizada de base para criação das classes `LexicalError` e `SemanticError`. É

responsável por instanciar erros e sua respectiva posição.

2. `LexicalError` – Classe que herda de `AnalysisError` e define um erro léxico, informando a mensagem de erro e sua respectiva posição.
3. SemanticError – Classe que herda de `AnalysisError` e define um erro semântico, informando a mensagem de erro e sua respectiva posição.
4. SyntaticError – Classe que herda de `AnalysisError` e define um erro sintático, informando a mensagem de erro e sua respectiva posição.
5. `ScannerConstants` – Classe responsável por definir as constantes de estado dos tokens e os tipos de erro que podem ser encontrados nos tokens.
6. `Constants` – Interface que herda de `ScannerConstants` e define identificadores inteiros para cada token.
7. `Token` – Classe que define os tokens encontrados na leitura do código. Possui como atributo um inteiro como identificador do token, uma *string* especificando o lexema e um inteiro com sua respectiva posição.
8. `Lexico` – Classe que implementa a interface `Constans` e é responsável por ler o código da linguagem ALGUMA e gerar os tokens encontrados ou erros.

4 TESTES DE EXECUÇÃO

Para utilizar as classes geradas pelo GALS, foi desenvolvida uma classe chamada `Principal`. Essa classe possui um método `main` que lê do usuário o nome de um arquivo com código em linguagem ALGUMA, lê este arquivo e cria um objeto do tipo `Lexico` passando o arquivo como parâmetro no construtor. Em um bloco `try-catch`, dentro de um laço de repetição, é chamado o método `nextToken()` do objeto da classe `Lexico`. Caso encontre novos tokens, imprime-se utilizando o método `toString()` da classe `Token`. Caso algum erro léxico seja encontrado, é lançada uma exceção que para a execução do laço e o erro é impresso.

Classe `Principal.java`

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Scanner;

public class Principal {

    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in);
        System.out.print("Nome do arquivo: ");
        String arquivo = in.next();
        InputStream is = new FileInputStream("ALGUMA/"+arquivo);
        InputStreamReader isr = new InputStreamReader(is);

        Lexico lexico = new Lexico(isr);
        try
        {
            Token t = null;
            while ( (t = lexico.nextToken()) != null )
            {
                System.out.println(t.toString());
            }
        }
        catch ( LexicalError e )
        {
            System.err.println("ERRO LÉXICO ( " + e.getMessage() + " ) @ " + e.getPosition());
        }
    }
}
```

4.1 Execução com código de fatorial

Algoritmo em ALGUMA para calcular o fatorial de N.

```
:DECLARACOES
n:INTEIRO
fatorial:INTEIRO

:ALGORITMO
LER n
ATRIBUIR n A fatorial
SE n = 0 ENTAO ATRIBUIR 1 A fatorial
ENQUANTO n > 1
  INICIO
    ATRIBUIR fatorial*(n-1) A fatorial
    ATRIBUIR n-1 A n
  FIM
IMPRIMIR fatorial
```

Execução do analisador:

```
Nome do arquivo: fatorial.alguma
18 ( : ) @ 0
2 ( DECLARACOES ) @ 1
21 ( n ) @ 13
18 ( : ) @ 14
4 ( INTEIRO ) @ 15
21 ( fatorial ) @ 23
18 ( : ) @ 31
4 ( INTEIRO ) @ 32
18 ( : ) @ 41
3 ( ALGORITMO ) @ 42
8 ( LER ) @ 52
21 ( n ) @ 56
6 ( ATRIBUIR ) @ 58
21 ( n ) @ 67
7 ( A ) @ 69
21 ( fatorial ) @ 71
10 ( SE ) @ 80
21 ( n ) @ 83
16 ( = ) @ 85
22 ( 0 ) @ 87
11 ( ENTAO ) @ 89
6 ( ATRIBUIR ) @ 95
22 ( 1 ) @ 104
7 ( A ) @ 106
21 ( fatorial ) @ 108
12 ( ENQUANTO ) @ 117
21 ( n ) @ 126
16 ( > ) @ 128
22 ( 1 ) @ 130
13 ( INICIO ) @ 134
6 ( ATRIBUIR ) @ 145
21 ( fatorial ) @ 154
15 ( * ) @ 162
19 ( ( ) @ 163
21 ( n ) @ 164
22 ( -1 ) @ 165
20 ( ) ) @ 167
7 ( A ) @ 169
21 ( fatorial ) @ 171
6 ( ATRIBUIR ) @ 184
21 ( n ) @ 193
22 ( -1 ) @ 194
7 ( A ) @ 197
21 ( n ) @ 199
14 ( FIM ) @ 203
9 ( IMPRIMIR ) @ 207
21 ( fatorial ) @ 216
```

4.2 Execução com código de soma dos números da sequência de Fibonacci

Algoritmo em ALGUMA para calcular a soma dos n primeiros números da sequência de Fibonacci.

```
:DECLARACOES
n:INTEIRO
fibonacci:INTEIRO
penultimo:INTEIRO
soma:INTEIRO
aux:INTEIRO

:ALGORITMO
LER n
ATRIBUIR 1 A fibonacci
ATRIBUIR 0 A penultimo
ATRIBUIR 1 A soma
ENQUANTO n > 0
  INICIO
    ATRIBUIR fibonacci A aux
    ATRIBUIR fibonacci+penultimo A fibonacci
    ATRIBUIR aux A penultimo
    ATRIBUIR soma+fibonacci A soma
    ATRIBUIR n-1 A n
  FIM
IMPRIMIR soma
```

Execução do analisador:

```
Nome do arquivo: somafibonacci.alguma
18 ( : ) @ 0
2 ( DECLARACOES ) @ 1
21 ( n ) @ 13
18 ( : ) @ 14
4 ( INTEIRO ) @ 15
21 ( fibonacci ) @ 23
18 ( : ) @ 32
4 ( INTEIRO ) @ 33
21 ( penultimo ) @ 41
18 ( : ) @ 50
4 ( INTEIRO ) @ 51
21 ( soma ) @ 59
18 ( : ) @ 63
4 ( INTEIRO ) @ 64
21 ( aux ) @ 72
18 ( : ) @ 75
4 ( INTEIRO ) @ 76
18 ( : ) @ 85
3 ( ALGORITMO ) @ 86
8 ( LER ) @ 96
21 ( n ) @ 100
6 ( ATRIBUIR ) @ 102
22 ( 1 ) @ 111
```

```
7 ( A ) @ 113
21 ( fibonacci ) @ 115
6 ( ATRIBUIR ) @ 125
22 ( 0 ) @ 134
7 ( A ) @ 136
21 ( penultimo ) @ 138
6 ( ATRIBUIR ) @ 148
22 ( 1 ) @ 157
7 ( A ) @ 159
21 ( soma ) @ 161
12 ( ENQUANTO ) @ 166
21 ( n ) @ 175
16 ( > ) @ 177
22 ( 0 ) @ 179
13 ( INICIO ) @ 183
6 ( ATRIBUIR ) @ 194
21 ( fibonacci ) @ 203
7 ( A ) @ 213
21 ( aux ) @ 215
6 ( ATRIBUIR ) @ 223
21 ( fibonacci ) @ 232
15 ( + ) @ 241
21 ( penultimo ) @ 242
7 ( A ) @ 252
21 ( fibonacci ) @ 254
6 ( ATRIBUIR ) @ 268
21 ( aux ) @ 277
7 ( A ) @ 281
21 ( penultimo ) @ 283
6 ( ATRIBUIR ) @ 297
21 ( soma ) @ 306
15 ( + ) @ 310
21 ( fibonacci ) @ 311
7 ( A ) @ 321
21 ( soma ) @ 323
6 ( ATRIBUIR ) @ 332
21 ( n ) @ 341
22 ( -1 ) @ 342
7 ( A ) @ 345
21 ( n ) @ 347
14 ( FIM ) @ 351
9 ( IMPRIMIR ) @ 355
21 ( soma ) @ 364
```

4.3 Execução de código com erro

Algoritmo em ALGUMA com erro (nome de variável não sendo iniciada com letra minúscula)

```
:DECLARACOES
n:INTEIRO
Fatorial:INTEIRO

:ALGORITMO
LER n
ATRIBUIR n A Fatorial
SE n = 0 ENTAO ATRIBUIR 1 A Fatorial
ENQUANTO n > 1
  INICIO
    ATRIBUIR Fatorial*(n-1) A Fatorial
    ATRIBUIR n-1 A n
  FIM
IMPRIMIR Fatorial
```

Execução do analisador:

```
Nome do arquivo: fatorial-com-erro.alguma
18 ( : ) @ 0
2 ( DECLARACOES ) @ 1
21 ( n ) @ 13
18 ( : ) @ 14
4 ( INTEIRO ) @ 15
ERRO LÉXICO ( Erro identificando FIM ) @ 23
```

Perceba que ao ler o `F` maiúsculo, da variável `Fibonacci`, o analisador entra no estado de reconhecimento do lexema `FIM`, pois é o primeiro lexema em que há a possibilidade de combinação. Porém, ao verificar o próximo caractere, não há mais combinações possíveis e o analisador retorna o erro léxico.

5 OUTROS GERADORES DE ANALISADOR LÉXICO

Além do GALS, pode-se citar como exemplo de ferramenta de geração de analisador léxico o Lex e o Flex. Diferentemente do GALS, essas ferramentas geram apenas os tokens que devem ser processados por outro programa que efetuará a análise sintática. O Lex foi desenvolvido para sistemas baseados em Unix, já o Flex para sistemas baseados em Linux. Além dessa, as duas ferramentas possuem outras diferenças: enquanto no Lex pode-se fornecer seu próprio código de

entrada e modificar o fluxo de caracteres, no Flex o usuário não possui essa opção. Também, é importante notar que o Flex é um sistema totalmente composto por software livre.