

Relatório REOBOY 2

Mateus Carvalho Gonçalves - 201810245

Após a criação do projeto como mostrado nas vídeo aulas, o primeiro passo foi a definição das Definições Regulares e dos Tokens. A definição da linguagem foi baseada na tabela de presente nos slides de aula “Análise Léxica parte 3”.

Definições Regulares:

letras : [a-zA-Z]

numeros : [0-9]

ws : [\n\s\r\t]

Tokens:

: {ws}

DEC : DECLARACOES

ALG : ALGORITMO

INT : INTEIRO

REA : REAL

ATR : ATRIBUIR

A : A

LER : LER

IMP : IMPRIMIR

SE : SE

ENT : ENTAO

ENQ : ENQUANTO

INI : INICIO

FIM : FIM

OpArit : (\+|\-|*|/)

OpRel : (<|<=|=|>=|>|<>)

OpBool : (E|OU)

Delim : :

AP : \((

FP : \)

Var : [a-z]({letras}|{numeros})*

NumI : {numeros}+

NumR : {numeros}+,{numeros}+

Str : \"({letras}|{numeros})*\"

Após isso, o código do analisador léxico foi gerado em Java e o arquivo contendo a função main foi cedido pelo professor para os testes. Os testes foram feitos parte na ferramenta “Simulador” presente na aplicação GALS (referente ao algoritmo “Fatorial de N”) e parte

utilizando os códigos compilando pelo IntelliJ Idea (referente ao algoritmo “Soma dos números da sequência de Fibonacci até N”). Os códigos usados foram os mesmos enviados na tarefa 2: “Análise Léxica Manual”. Foram provocados erros léxicos de propósito nos dos casos.

Testes:

1 - Fatorial de N

a. Utilizando algoritmo correto

The screenshot shows the GALVS - Gerador de Análise de Sintaxe (GALS) interface. The main window displays the source code for a factorial algorithm. The left pane shows the regular expressions defined for the lexer: `letras : [a-zA-Z]`, `numeros : [0-9]`, and `ws : [\n\s\r\t]`. The main pane shows the code being analyzed, which is a correct implementation of a factorial algorithm. The right pane shows the output of the lexical analysis, listing tokens and their positions.

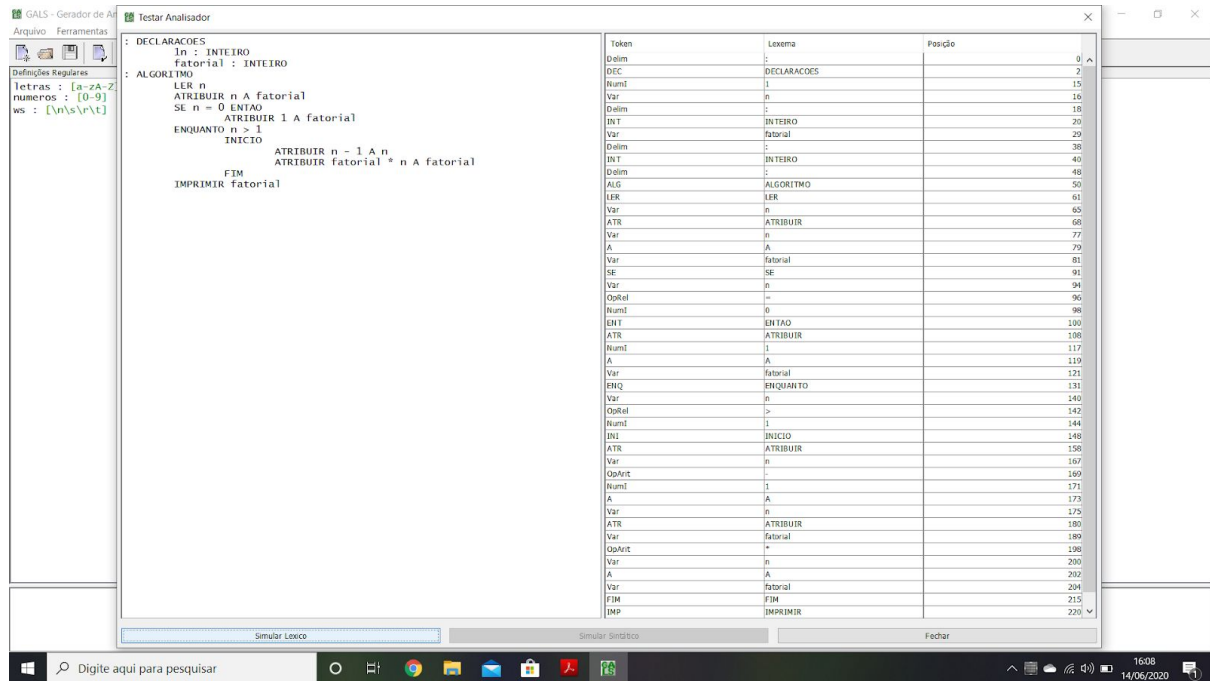
Token	Lexema	Posição
Delim	:	6
DEC	DECLARACOES	7
Var	n	15
Delim	:	17
INT	INTEIRO	19
Var	fatorial	28
Delim	:	37
INT	INTEIRO	39
Delim	:	47
ALG	ALGORITMO	49
LER	LER	60
Var	n	64
ATR	ATRIBUIR	67
Var	n	76
A	A	78
Var	fatorial	80
SE	SE	90
Var	n	93
OpRel	=	95
NumI	0	97
ENT	ENTAO	99
ATR	ATRIBUIR	107
NumI	1	116
A	A	118
Var	fatorial	120
ENQ	ENQUANTO	130
Var	n	139
OpRel	>	141
NumI	1	143
INT	INICIO	147
ATR	ATRIBUIR	157
Var	n	166
OpArit	-	168
NumI	1	170
A	A	172
Var	n	174
ATR	ATRIBUIR	179
Var	fatorial	188
OpArit	*	197
Var	n	199
A	A	201
Var	fatorial	203
FIM	FIM	214
IMP	IMPRIMIR	219
Var	fatorial	226

b. Aponta erro léxico: uso de palavra não definida “SENAO”

The screenshot shows the GALVS - Gerador de Análise de Sintaxe (GALS) interface. The main window displays the source code for a factorial algorithm, but with a typo: `SENAO` instead of `SENÃO`. The left pane shows the regular expressions defined for the lexer: `letras : [a-zA-Z]`, `numeros : [0-9]`, and `ws : [\n\s\r\t]`. The main pane shows the code being analyzed. The right pane shows the output of the lexical analysis, listing tokens and their positions. The error is highlighted in the output table.

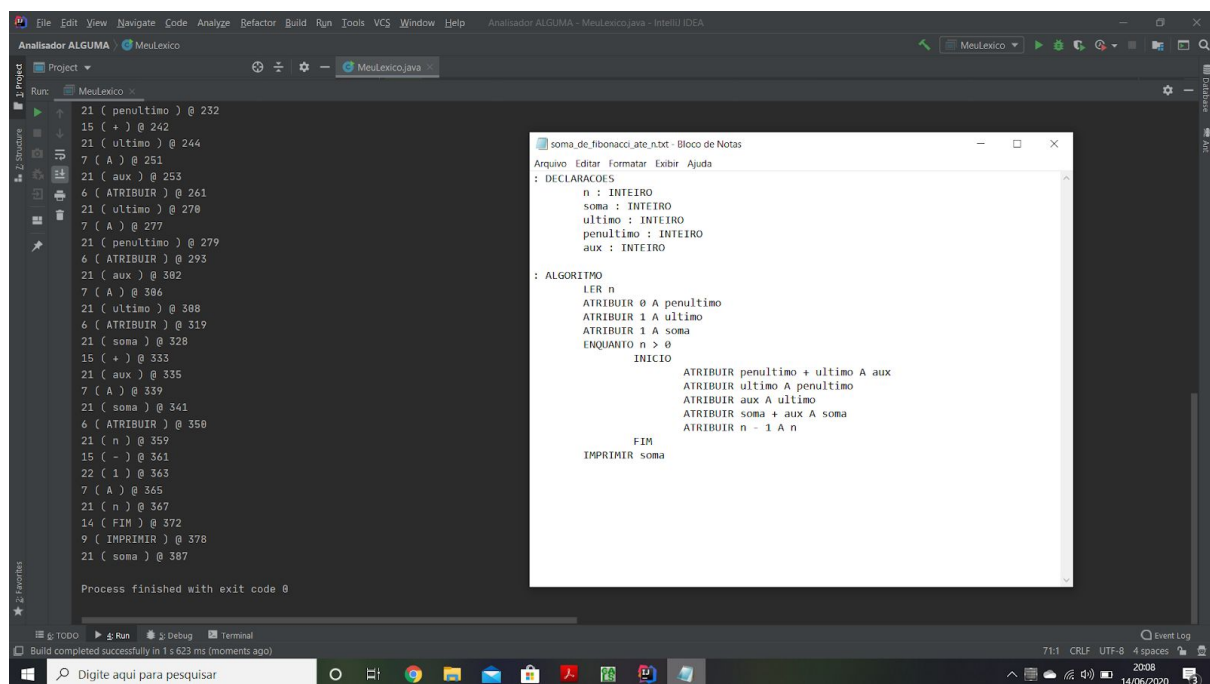
Token	Lexema	Posição
Delim	:	6
DEC	DECLARACOES	7
Var	n	15
Delim	:	17
INT	INTEIRO	19
Var	fatorial	28
Delim	:	37
INT	INTEIRO	39
Delim	:	47
ALG	ALGORITMO	49
LER	LER	60
Var	n	64
ATR	ATRIBUIR	67
Var	n	76
A	A	78
Var	fatorial	80
SE	SE	90
Var	n	93
OpRel	=	95
NumI	0	97
ENT	ENTAO	99
ATR	ATRIBUIR	107
NumI	1	116
A	A	118
Var	fatorial	120
SE	SE	130
ERRO LÉXICO	Caractere não esperado	132

- c. Nesse caso, há um erro léxico na variável “1n” que não foi pego pelo analisador, ao invés disso, ele resolveu como dois tokens diferente, um NumI e um Var. Sendo assim, esse erro teria que ser tratado ou na etapa de análise sintática ou poderia incluir algumas regras nas expressões regulares de inclusão de espaços ou quebras de linha no final das leituras das palavras, mas esse culminaria em uma programação menos flexível dos algoritmos

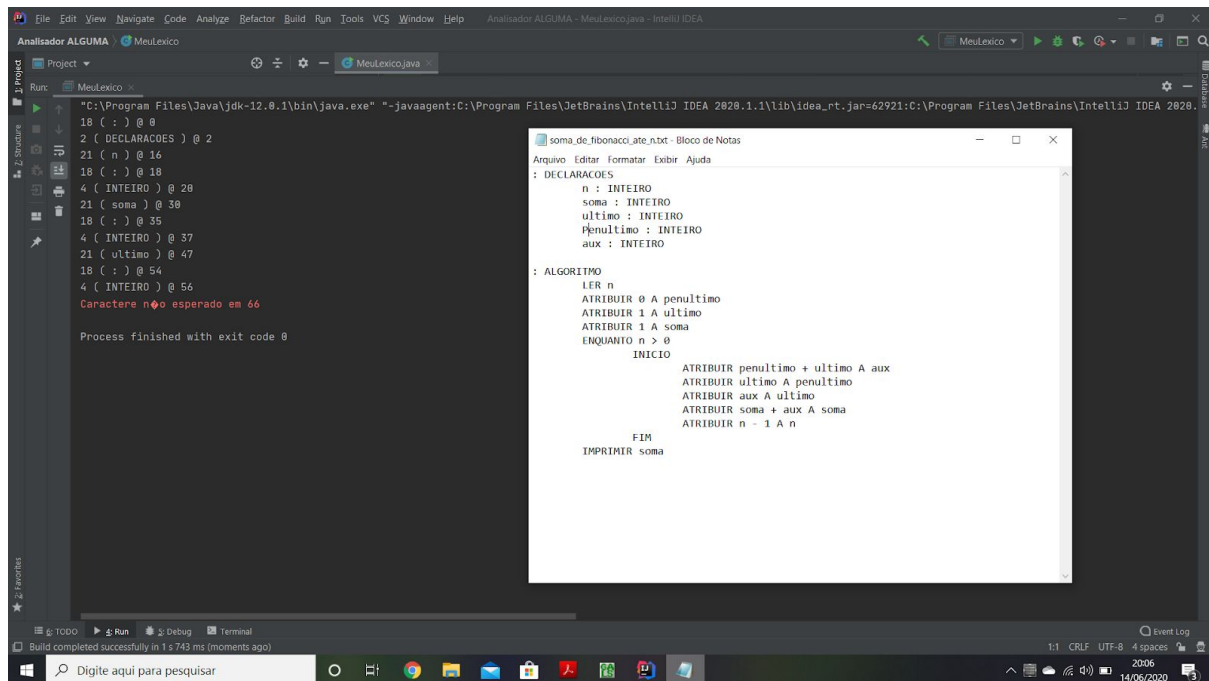


2 - Soma dos números da sequência de Fibonacci até N

a. Utilizando o algoritmo correto



- b. Penultimo não atende a nenhuma expressão regular e fere a regra de começar com letras minúsculas do identificador Var



The screenshot shows the IntelliJ IDEA interface. The main editor displays a Java program that has been executed successfully. The output window shows the following text:

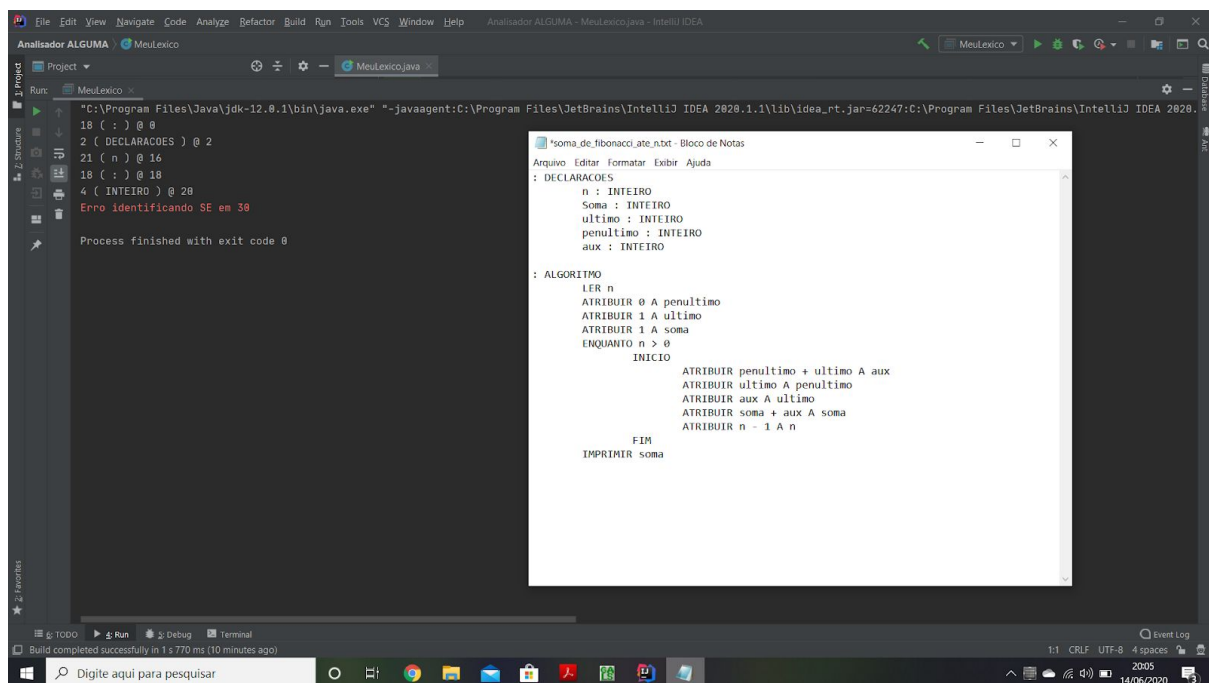
```
"C:\Program Files\Java\jdk-12.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.1.1\lib\idea_rt.jar=62921:C:\Program Files\JetBrains\IntelliJ IDEA 2020.1.1\bin" -Dfile.encoding=UTF-8
18 ( : ) @ 0
2 ( DECLARACOES ) @ 2
21 ( n ) @ 16
18 ( : ) @ 18
4 ( INTEIRO ) @ 20
21 ( soma ) @ 30
18 ( : ) @ 35
4 ( INTEIRO ) @ 37
21 ( ultimo ) @ 47
18 ( : ) @ 54
4 ( INTEIRO ) @ 56
Caractere n não esperado em 66
Process finished with exit code 0
```

Overlaid on the IDE is a Notepad window titled "soma.de.fibonacci.ate.txt - Bloco de Notas". It contains a Pseudocode document with the following content:

```
Arquivo Editar Formatar Exibir Ajuda
: DECLARACOES
n : INTEIRO
soma : INTEIRO
ultimo : INTEIRO
penultimo : INTEIRO
aux : INTEIRO

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

- c. Como acima, uma palavra para Var deveria começar com letra minúscula. O interessante da saída é que o analisador aponta que esperava um SE



The screenshot shows the IntelliJ IDEA interface. The main editor displays a Java program that has been executed successfully. The output window shows the following text:

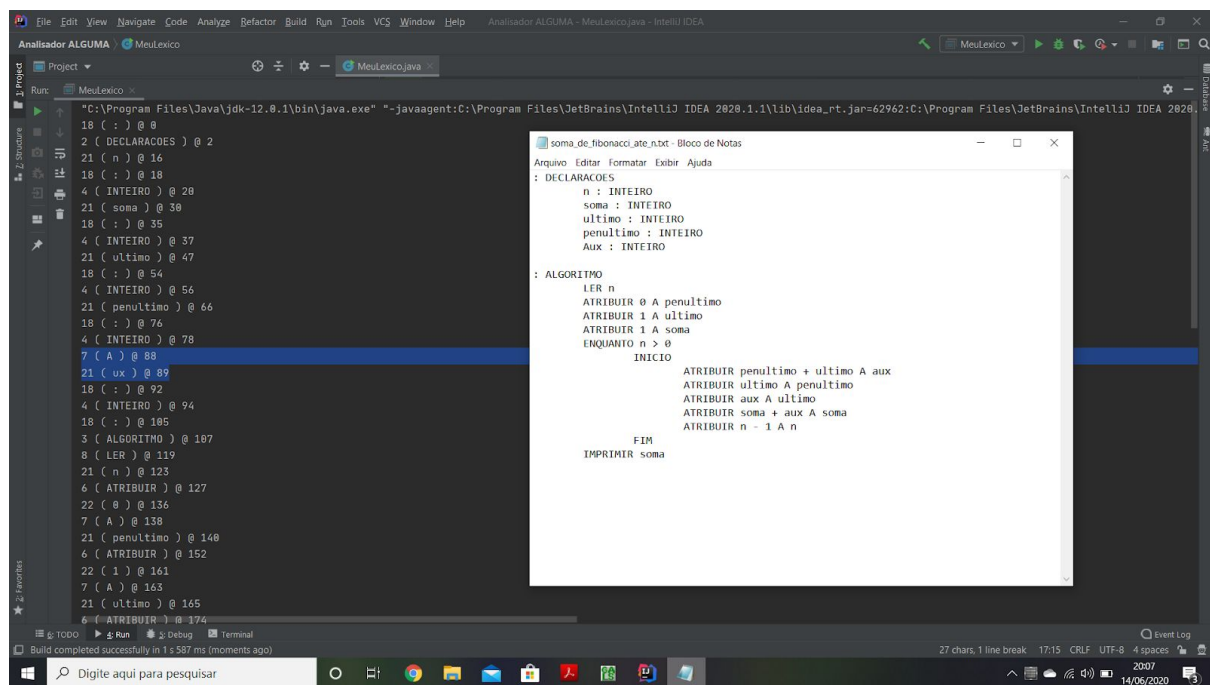
```
"C:\Program Files\Java\jdk-12.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.1.1\lib\idea_rt.jar=62247:C:\Program Files\JetBrains\IntelliJ IDEA 2020.1.1\bin" -Dfile.encoding=UTF-8
18 ( : ) @ 0
2 ( DECLARACOES ) @ 2
21 ( n ) @ 16
18 ( : ) @ 18
4 ( INTEIRO ) @ 20
Erro identificando SE em 30
Process finished with exit code 0
```

Overlaid on the IDE is a Notepad window titled "soma.de.fibonacci.ate.txt - Bloco de Notas". It contains a Pseudocode document with the following content:

```
Arquivo Editar Formatar Exibir Ajuda
: DECLARACOES
n : INTEIRO
Soma : INTEIRO
ultimo : INTEIRO
penultimo : INTEIRO
aux : INTEIRO

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

d. Mesmo problema apontado na 1c, analisador avalia “A” e “ux” ao invés de Aux.



Outros geradores de analisadores léxicos

ANTLR: A partir de uma definição de gramática por expressões regulares é possível gerar um lexer (scanner) e um parser em diversas linguagens de programação diferentes, como Java, Python, C#, Javascript, C++, entre outros. Invocando eles em um texto, é gerado uma Abstract Syntax Tree (AST) em memória, uma representação das estruturas encontradas. É amplamente usado tanto para fins de aprendizado quanto no mercado, sendo utilizado por aplicações como Twitter e Netbeans.

FLEX: Fast LEXical analyzer generator é uma ferramenta que gera apenas um lexer a partir da definição de uma gramática. Um arquivo FLEX é constituído por 3 seções delimitadas pelo separador `%%`. A primeira é onde se faz definições (como as definições regulares da ferramenta GALS), a segunda seção é utilizada para as regras do FLEX propriamente ditas, ou seja, as expressões regulares para geração dos tokens, e a terceira para as rotinas adicionais do usuário.

Pelo que eu entendi essas ferramentas são utilizadas pelo terminal e cada uma tem uma forma diferente de especificar as definições.