[**Trabalho 1 -**](https://moodle.joinville.udesc.br/mod/resource/view.php?id=110242) **Análise Léxica**

**Gabriel Anselmo Ramos e Bruno Rafael dos Santos**

**1. Definição da gramática**

<atribuição> ::= <variável> ":=" <expressão>

<bool-lit> ::= true | false

<chamada-de-função> ::= <id> "(" <lista-de-expressões> | <vazio> ")"

<chamada-de-procedimento> ::= <id> "(" <lista-de-expressões> | <vazio> ")"

<comando> ::= <atribuição> | <condicional> | <iterativo> | <chamada-de-procedimento> | <comando-composto>

<comando-composto> ::= begin <lista-de-comandos> end

<condicional> ::= if <expressão> then <comando> ( else <comando> | <vazio> )

<corpo> ::= <declarações> <comando-composto>

<declaração> ::= <declaração-de-variável> | <declaração-de-função> | <declaração-de-procedimento>

<declaração-de-função> ::= function <id> "(" <lista-de-parâmetros> | <vazio> ")" : <tipo-simples> ; <corpo>

<declaração-de-procedimento> ::= procedure <id> "(" <lista-de-parâmetros> | <vazio> ")" ; <corpo>

<declaração-de-variável> ::= var <lista-de-ids> : <tipo>

<declarações> ::= <declaração> ; | <declarações> <declaração> ; | <vazio>

<digito> ::= 0 | 1 | 2 | ... | 9

<expressão> ::= <expressão-simples> | <expressão-simples> <op-rel> <expressão-simples>

<expressão-simples> ::= <expressão-simples> <op-ad> <termo> | <termo>

<fator> ::= <variável> | <literal> | "(" <expressão> ")" | <chamada-de-função>

<float-lit> ::= <int-lit> . <int-lit> | <int-lit> . | . <int-lit>

<id> ::= <letra> | <id> <letra> | <id> <digito>

<int-lit> ::= <digito> | <int-lit> <digito>

<iterativo> ::= while <expressão> do <comando>

<letra> ::= a | b | c | ... | z

<lista-de-comandos> ::= <comando> ; | <lista-de-comandos> <comando> ; | <vazio>

<lista-de-expressões> ::= <lista-de-expressões> , <expressão> | <expressão>

<lista-de-ids> ::= <id> | <lista-de-ids> , <id>

<lista-de-parâmetros> ::= <parâmetros> | <lista-de-parâmetros> ; <parâmetros>

<literal> ::= <bool-lit> | <int-lit> | <float-lit>

<op-ad> ::= + | - | or

<op-mul> ::= \* | / | and

<op-rel> ::= < | > | <= | >= | = | <>

<outros> ::= ! | @ | # | ...

<parâmetros> ::= ( var | <vazio> ) <lista-de-ids> : <tipo-simples>

<programa> ::= program <id> ; <corpo> .

<seletor> ::= <seletor> "[" <expressão> "]" | "[" <expressão> "]" | <vazio>

<termo> ::= <termo> <op-mul> <fator> | <fator>

<tipo> ::= <tipo-agregado> | <tipo-simples>

<tipo-agregado> ::= array ( "[" <literal> .. <literal> "]" ) of <tipo>

<tipo-simples> ::= integer | real | boolean

<variável> ::= <id> <seletor>

<vazio> ::= ε

**2. Especificação FLEX**

DIGITO ([0-9])

BOOL-LIT true | false

OP-AD “+” | “-” | “or”

OP-MUL “\*” | “/” | “and”

OP-REL “<” | “>” | “<=” | “>=” | “=” | ”<>”

OUTROS ”!” | “@” | “#” | “$” | “%” | “&”

TIPO “integer“ | “real“ | “boolean“

VAZIO “ε“

LETRA [a-z]

Para as especificacao no flex fizemos a definicão dos termos terminas e definimos as regras seguintes a partir da sessão de definições.

**3. Comentários**

Definimos a tabela de símbolos salvando os tokens em uma pilha, onde podemos fazer a alocação dinamica de memoria quando necessario, usando uma struct para guardar o token, a linha e coluna onde ele se encontra no arquivo e um ponteiro para apontar para o próximo token da tabela de símbolos. Como apenas pensamos em salvar os tokens no memento não definimos a estrutura de dados que vamos realmente usar no nosso trabalho pra as demais fase do compilador, logo quando tivermos que verificar e fazer buscas na tabela de simbolos eu vejo que seria mais interessante usar um função hash para que as consultas sejam mais rapida durante a compilação.