UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Relatório - AL 1

Construção de Analisador Léxico

Artur Ribeiro Alfa [17103919]

Augusto Vieira Coelho Rodrigues [19100517]

Leonardo Vieira Nunes [19102923]

Thainan Vieira Junckes [19100545]

FLORIANÓPOLIS, 15 DE MAIO DE 2022

1. Identificação dos tokens

- Operadores Aritméticos
    - PLUS -> adição
    - MINUS -> subtração
    - TIMES -> multiplicação
    - DIVIDE -> divisão
    - MOD -> resto da divisão
    - ASSIGN -> igual

- Operadores Relacionais
    - LT -> less than, menor que
    - GT -> greater than, maior que
    - LTE -> less than or equal to, menor ou igual que
    - GTE -> greater than or equal to, maior ou igual que
    - EQUAL -> Igualdade
    - INEQUAL -> Inigualdade

- Operadores para cada pontuação
    - DOT -> ponto
    - COMMA -> vírgula
    - SEMICOLON -> ponto e vírgula
    - RIGHTPARENTHESES -> parênteses direito
    - LEFTPARENTHESES -> parênteses esquerdo
    - RIGHTBRACKET -> colchete direito
    - LEFTBRACKET -> colchete esquerdo
    - RIGHTBRACE -> chave direita
    - LEFTBRACE -> chave esquerda

- Constantes
    - INTCONSTANT -> constante com número inteiro
    - FLOATCONSTANT -> constante com número em ponto flutuante
    - STRINGCONSTANT -> constante string
    - NULL > valor nulo

- Palavras reservadas
    - DEF -> definição de função
    - IDENT -> identificação
    - INT -> inteiro
    - FLOAT -> ponto flutuante
    - STRING -> string
    - BREAK -> interrompe um laço de repetição
    - PRINT -> imprime um valor na tela
    - READ -> lê um valor
    - RETURN -> retorna um valor
    - IF -> condição “se”
    - ELSE -> usada em conjunto com o IF, condição “senão”

    - FOR -> For loop, repete um código até que a condição seja atingida
    - NEW -> aloca memória para um valor

- Outros
    - IGNORE -> Ignora espaços e tabs
    - LINEBREAK -> Quebra de linha

2. Definições regulares para cada token

- Operadores Aritméticos
    - PLUS -> +
    - MINUS -> -
    - TIMES -> *
    - DIVIDE -> /
    - MOD -> %
    - ASSIGN -> =

- Operadores Relacionais
    - LT -> <
    - GT -> >
    - LTE -> <=
    - GTE -> >=
    - EQUAL -> ==
    - INEQUAL -> !=

- Operadores para cada pontuação
    - DOT -> .
    - COMMA -> ,
    - SEMICOLON -> ;
    - RIGHTPARENTHESES ->)
    - LEFTPARENTHESES -> (
    - RIGHTBRACKET ->]
    - LEFTBRACKET -> [
    - RIGHTBRACE -> }
    - LEFTBRACE -> {

- Não-triviais (notação em regex)
    - INTCONSTANT -> r'\d+'
    - FLOATCONSTANT -> r'\d+\.\d+'
    - STRINGCONSTANT -> r'"[^\n"\r]*"'
    - IDENT -> r'[a-zA-Z_][a-zA-Z_0-9]*'

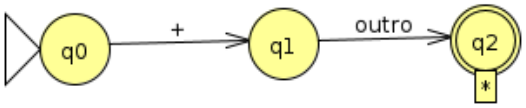
- Palavras reservadas
    - DEF -> def
    - INT -> int
    - FLOAT -> float
    - STRING -> string

    - BREAK -> break
    - PRINT -> print
    - READ -> read
    - RETURN -> return
    - IF -> if
    - ELSE -> else
    - FOR -> for
    - NEW -> new
    - NULL > null

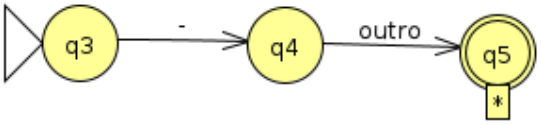
- Outros
    - IGNORE -> ' \t'
    - LINEBREAK -> r'\n+'

3. Diagramas de transição para cada token

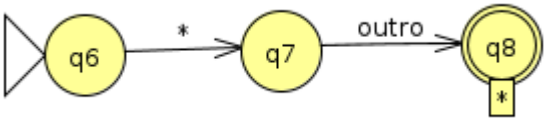
PLUS:



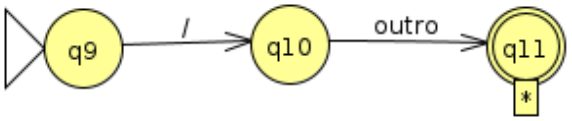
MINUS:



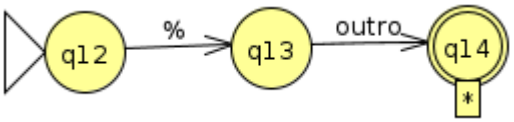
TIMES:



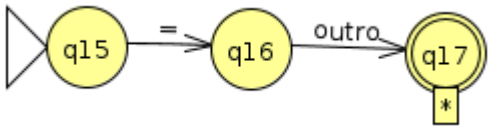
DIVIDE:



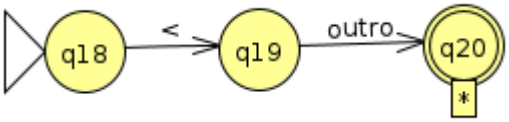
MOD:



ASSIGN:

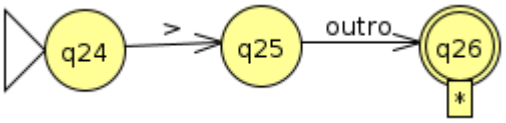


LT:

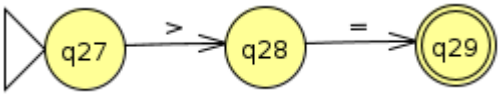


LTE:

GT:



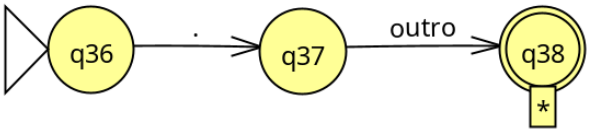
GTE:



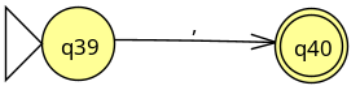
EQUAL:

INEQUAL:

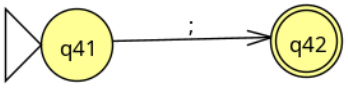
DOT:



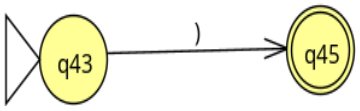
COMMA:



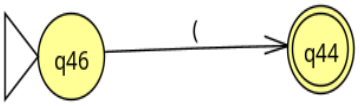
SEMICOLON:



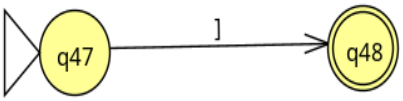
RIGHTPARENTHESES:



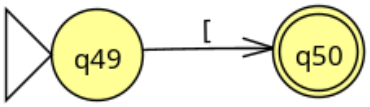
LEFTPARENTHESES:



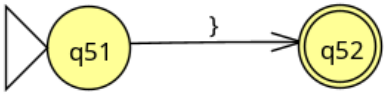
RIGHTBRACKET:



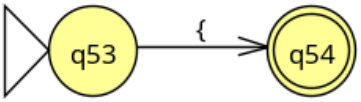
LEFTBRACKET:



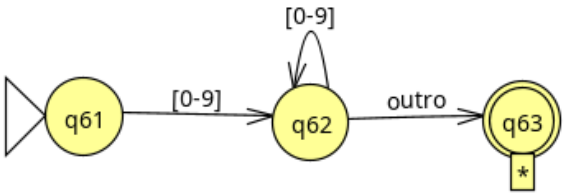
RIGHTBRACE:



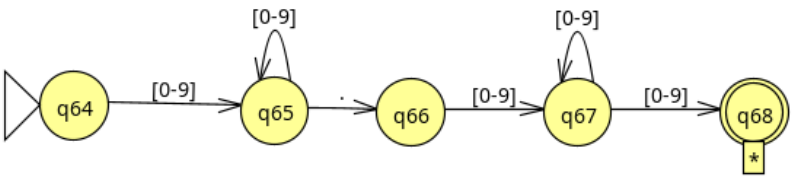
LEFTBRACE:



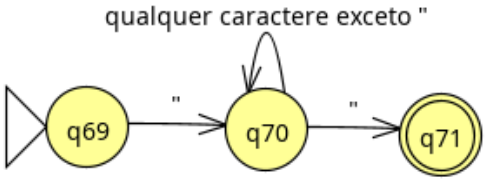
INTCONSTANT:



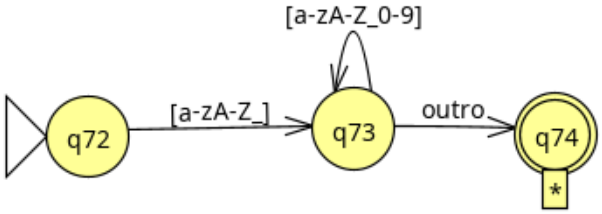
FLOATCONSTANT:



STRINGCONSTANT:



IDENT:



4. Tabela de símbolos

Para implementar a tabela de símbolos, utilizamos um dicionário do python em que a chave é o identificador, assim armazenando os identificadores (ou variáveis) do programa.
Abaixo podemos visualizar os atributos que são armazenados na tabela ao executar a função *print_symbols_table*. Os atributos escolhidos são o tipo da variável (TIPO), a linha em que a variável foi declarada (DECLARAÇÃO), assim como a linha em que ela é referenciada (REFERENCIADO). Se ela é referenciada mais de uma vez em uma mesma linha, a coluna REFERENCIADO irá conter valores repetidos.

```
========= TABELA DE SÍMBOLOS =========
+------+-------+--------------------+-------------------------------------------------------------+
| NOME |  TIPO | DECLARAÇÃO (linha) |                   REFERENCIADO (linha)                      |
+------+-------+--------------------+-------------------------------------------------------------+
|  x   | float |         3          |                [7, 10, 11, 11, 12, 13]                      |
|  z   | float |         4          |                        [12, 13]                             |
|  i   |  int  |         24         | [9, 9, 9, 9, 27, 27, 27, 28, 29, 35, 35, 35, 35, 36]        |
| max  |  int  |         6          |                          [8, 9]                             |
|  y   |  int  |         22         |                      [25, 29, 36]                           |
|  j   |  int  |         23         |                     [26, 29, 30, 30]                        |
+------+-------+--------------------+-------------------------------------------------------------+
```

Se uma variável não foi declarada e/ou não é do tipo def, string, int ou float; ela terá no dicionário os valores NAO_DECLARADO e/ou SEM_TIPO ao invés de conter seu respectivo tipo e linha declarada.

5. Ferramenta PLY

Para desenvolver este trabalho, utilizamos a ferramenta PLY (Python Lex-Yacc), que basicamente permite dividir o texto de entrada em uma coleção de tokens ao especificar uma coleção de regras de expressão regular, além de outras funções.
A implementação da ferramenta consiste na criação de uma lista de tokens, atribuir uma expressão regular para cada token e criar um dicionário para as palavras reservadas (chave e valor tendo palavra e token respectivamente). A seguir, um pequeno exemplo:

```
# tokens list
tokens = ['PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'ASSIGN']
```

```
#regular expression
t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
t_DIVIDE = r'/'
t_ASSIGN = r'='

#reserved words
reserved = {
            'int': 'INT',
            'float': 'FLOAT',
            'string': 'STRING',
            'new': 'NEW',
            'def': 'DEF',
            'break': 'BREAK',
            'print': 'PRINT',
            'return': 'RETURN',
            'read': 'READ',
            'if': 'IF',
            'else': 'ELSE',
            'for': 'FOR',
            'null': 'NULL'
            }
```

O nosso programa tem como entrada um arquivo .lcc contendo o texto a ser analisado e como saída a seguinte sequência:

1. Erros léxicos. Contendo qual caracter, linha e coluna causou o erro;
2. Uma tabela de tokens, contendo o token, o caracter (ou sequência de caracteres) correspondente, e sua linha e coluna no texto;
3. E por último, a tabela de símbolos, explicada na seção 4 deste relatório.

Exemplo:

```
----------------------------------------
ERRO LÉXICO
Caracter: '|'
Linha: 9
Coluna: 11
----------------------------------------
```

========= TABELA DE TOKENS =========

TOKEN	VALOR	LINHA	COLUNA
DEF	def	1	1
IDENT	principal	1	5
LEFTPARENTHESES	(	1	14
RIGHTPARENTHESES	)	1	15
LEFTBRACE	{	2	1
INT	int	3	3
IDENT	C	3	7
SEMICOLON	;	3	8
INT	int	4	3
IDENT	D	4	7
SEMICOLON	;	4	8
INT	int	5	3
IDENT	R	5	7
SEMICOLON	;	5	8
IDENT	C	6	3
ASSIGN	=	6	5
INTCONSTANT	4	6	7
SEMICOLON	;	6	8
IDENT	D	7	3
ASSIGN	=	7	5
INTCONSTANT	5	7	7
SEMICOLON	;	7	8
IDENT	R	8	3
ASSIGN	=	8	5
IDENT	func1	8	7
LEFTPARENTHESES	(	8	12
IDENT	C	8	13
COMMA	,	8	14
IDENT	D	8	16
RIGHTPARENTHESES	)	8	17
SEMICOLON	;	8	18
RETURN	return	9	3
SEMICOLON	;	9	9
RIGHTBRACE	}	10	1

========= TABELA DE SÍMBOLOS =========

NOME	TIPO	DECLARAÇÃO (linha)	REFERENCIADO (linha)
principal	def	1	[]
C	int	3	[6, 8]
D	int	4	[7, 8]
R	int	5	[8]
func1	SEM_TIPO	NAO_DECLARADO	[8]