

Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática
Otimização de Sistemas

Documentação:
Compilador linguagem L

Iago Augusto Coelho Morgado
Homenique Vieira Martins
Guilherme Cósso Lima Pimenta

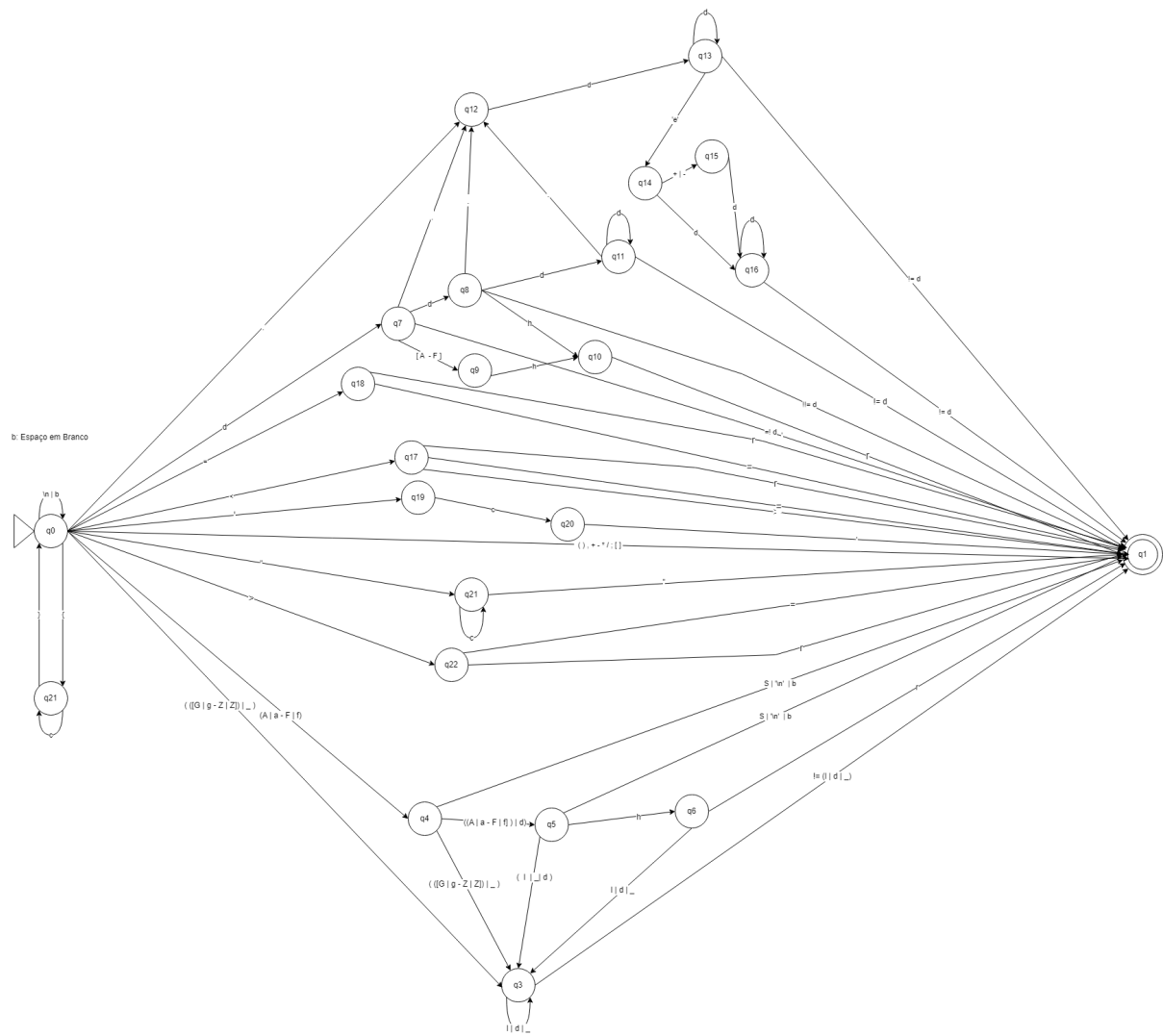
Tabela de Token e Lexemas

Nº	Token	Lexema
1	final	'f' 'i' 'n' 'a' 'l'
2	integer	'i' 'n' 't' 'e' 'g' 'e' 'r'
3	char	'c' 'h' 'a' 'r'
4	real	'r' 'e' 'a' 'l'
5	string	's' 't' 'r' 'i' 'n' 'g'
6	boolean	'b' 'o' 'o' 'l' 'e' 'a' 'n'
7	for	'f' 'o' 'r'
8	if	'i' 'f'
9	else	'e' 'l' 's' 'e'
10	and	'a' 'n' 'd'
11	or	'o' 'r'
12	not	'n' 'o' 't'
13	==	'=' '='
14	=	'='
15	>	'>'
16	>=	'>' '='
17	<	'<'
18	<=	'<' '='
19	<>	'<' '>'
20	('('
21)	')'
22	['['
23]	']'
24	,	','
25	+	'+'

26	-	'_'
27	div	('d' 'i' 'v' '/')
28	*	'*'
29	mod	'm' 'o' 'd'
30	const	((HEX D) (HEX D) 'h') ∪ ((D *. D + [e [- +] D +]) D +) ∪ (' [C] ') ∪ (" C * "
31	id	(L _)(L D _)*
32	;	'.'
33	:	'.'
34	false	'f' 'a' 'l' 's' 'e'
35	true	't' 'r' 'u' 'e'
36	begin	'b' 'e' 'g' 'i' 'n'
37	end	'e' 'n' 'd'
38	write	'w' 'r' 'i' 't' 'e'
49	writeln	'w' 'r' 'i' 't' 'e' 'l' 'n'
40	readln	'r' 'e' 'a' 'd' 'l' 'n'
41	fim_arquivo	'eof'

Legenda	
C	qualquer caractere aceito pela linguagem (letra, dígito , espaço, sublinhado, etc)
L	qualquer letra do alfabeto
D	qualquer dígito de 0 a 9
HEX	qualquer caractere do alfabeto de a até f (a,b,c,d,e,f), característico de números hexadecimais

Analizador Léxico



Gramática

S -> { **D** } { **Cmd** } fim_arquivo

//Declarações

D -> **D1 C** { , **C1** } ; | **D2**

***C** -> **id** [(= [-] **const**) | ([**const**])]

D1 -> (**char** | **integer** | **real** | **boolean** | **string**)

D2 -> **final id** = [-] **const** ;

//Comandos

Cmd -> (([**A** | **L** | **E**] ;) | (**R** | **T**))

Cmd1 -> (**A** | **R** | **T** | **L** | **E**)

A -> **id** [[**const**]] = **Exp**

R -> **for** (**R1** ; **Exp** ; **R1**) **T1**

***R1** -> [**Cmd1**] { , **Cmd1** }

T -> **if** (**Exp**) **T1** [**else T1**]

T1 -> (**Cmd** | **begin** { **Cmd** } **end**)

L -> **readln** (**id**)

E -> (**write** | **writeln**) (**E1**)

E1 -> **Exp** { , **Exp** }

//Expressão

Exp -> **Exp1** { (== | < | <= | > | >= | <>) **Exp1** }

Exp1 -> [-] **Exp2** { (+ | - | **or**) **Exp2** }

Exp2 -> **Exp3** { (* | **mod** | (**div** | /) | **and**) **Exp3** }

Exp3 -> **Exp4** | **not** **Exp4**

Exp4 -> **Exp5** | **real**(**Exp5**) | **integer**(**Exp5**)

Exp5 -> **const** | **id** [[**const**]] | (**Exp**)

Legenda	
Azul	Símbolo Terminal
Vermelho	Símbolo não terminal

Esquema de Tradução

*S -> { D } { Cmd } fim_arquivo	Regras de Tradução
<u>DECLARAÇÕES:</u> D -> D1 C { , C' } ; final id [1] = [- [2]] const [20]; D1 -> (char integer real boolean string) [4] C -> id [5] [(= [- [2]] const [3]) ([const [6]])]	[1, 2, 20] [4] [5, 2, 3, 6]
<u>COMANDOS :</u> Cmd -> (([A L E] ;) (R T)) *Cmd1 -> (A R T L E) A -> id [7] [[const [8]]] = Exp ¹ [9] R -> for (R1 ; Exp[10] ; R1') T1 *R1 -> [Cmd1] { , Cmd1 } T -> if (Exp [10]) T1 [else T1] T1 -> (Cmd begin { Cmd } end) L -> readln (id [7][11]) E -> (write writeln) (E1) E1 -> Exp ⁰ [11] { , Exp ⁰ [11] }	[7, 8, 9] [10] [10] [7, 11] [11]
<u>Expressão :</u> Exp0 -> Exp1 [12] { (== < <= > >= <>) [13] } Exp1' [15] Exp1 -> [- [2]] Exp2 [12] { (+ - or) [13] } Exp2' [16] Exp2 -> Exp3 [12] { (* mod (div /) and) [13] } Exp3' [17] Exp3 -> Exp4 [12] not (Exp4 [12][18]) Exp4 -> Exp5 [12] real(Exp5 [12][19]) integer(Exp5 [12][19]) *Exp5 -> const [14] id [7] [[const [8]]] (Exp)	[12, 13 ,15] [2, 12 ,13 ,16] [12, 13, 17] [12, 18] [12 , 19] [14, 7, 8]

Legenda	
Azul	Símbolo Terminal
Vermelho	Símbolo não terminal
Negrito	Ação Semântica

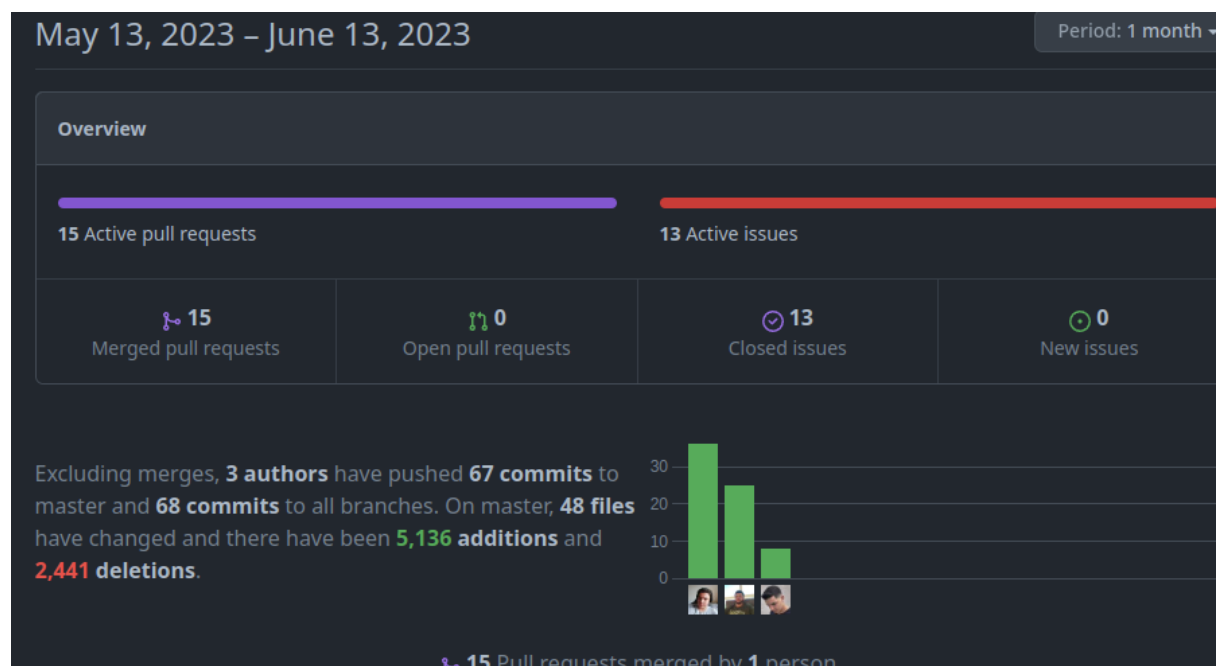
Regra	Ação semântica
[1]	<pre> { Se Identificador.Class != TOKEN_CLASS_UNDEFINED ERROR (IDENTIFIER_ALREADY_DECLARED) ; Senão Identificador.Class = TOKEN_CLASS_CONSTANT ; } </pre>
[2]	<pre> { Flag negativo = True } </pre>
[3]	<pre> { Se Negativo == True && !(Const.Tipo == Inteiro Const.Tipo == Real); ERROR (INCOMPATIBLE_TYPES) ; Senão Se (Identificador.tipo != Const.tipo &&(Identificador.tipo == TOKEN_TYPE_REAL && Const.tipo != TOKEN_TYPE_INTEGER)); ERROR (INCOMPATIBLE_TYPES) ; Senão Identificador.Tipo = Const.Tipo; } </pre>
[4]	<pre> { D1.Tipo = Token.Tipo } </pre>
[5]	<pre> { Se Identificador.Class != TOKEN_CLASS_UNDEFINED ERROR (IDENTIFIER_ALREADY_DECLARED) ; Senão Identificador.Tipo = D1.Tipo Identificador.Class = TOKEN_CLASS_VARIABLE ; } </pre>
[6]	<pre> { Se Identificador.Class != TOKEN_CLASS_UNDEFINED ERROR (IDENTIFIER_ALREADY_DECLARED) ; Se Identificador.type != TOKEN_TYPE_INTEGER ERROR (INCOMPATIBLE_TYPES) ; Senão Identificador.maxT = const.lex Identificador.Class = TOKEN_CLASS_VETOR; } </pre>

[7]	Se Identificador.Class == <code>TOKEN_CLASS_UNDEFINED</code> <code>ERROR (IDENTIFIER_NO_DECLARED) ;</code> Se Identificador.Class == <code>TOKEN_CLASS_CONSTANT</code> <code>ERROR (MISMATCHED_IDENTIFIER_CLASS) ;</code>
[8]	{ Se Identificador.tipo != <code>TOKEN_TYPE_INTEGER</code> ; <code>ERROR (INCOMPATIBLE_TYPES) ;</code> Senão Se Identificador.maxT <= const.lex <code>ERROR (IDENTIFIER_NO_DECLARED) ;</code> Senão <code>FLAG.VET = TRUE;</code> }
[9]	{ se Identificador.tipo != Exp ¹ .tipo && !(id.tipo == real & Exp ¹ .tipo == inteiro) && !(id.tipo == char & Exp ¹ .tipo == String) <code>ERROR (INCOMPATIBLE_TYPES) ;</code> }
[10]	{ se Exp.tipo != <code>TOKEN_TYPE_BOOLEAN</code> <code>ERROR (INCOMPATIBLE_TYPES) ;</code> }
[11]	{ se Identificador.tipo != <code>TOKEN_TYPE_BOOLEAN</code> <code>ERROR (INCOMPATIBLE_TYPES) ;</code> }
[12]	Obs: X = NÚMERO INTEIRO DE 0 A 4 Número da expressão) { <code>EXP(x).tipo = EXP(x+1).tipo</code> <code>EXP(x).end = EXP(x+1).end</code> }
[13]	{ <code>EXP.operator = token.lex</code> }
[14]	{ <code>Exp5.tipo = const.tipo</code> }
[15]	{ se ¹ Exp.tipo != Exp1.Tipo se ² !((Exp.tipo = real ou Exp.tipo = inteiro) &&

	<pre> (Exp1.tipo = real ou Exp1.tipo = inteiro)) ERROR (INCOMPATIBLE_TYPES) ; senao¹ se³ Exp.tipo == "String" & Exp.operator != "==" ERROR (INCOMPATIBLE_TYPES) ; } </pre>
[16]	<pre> { se Exp1.operator == or se !(Exp1.tipo == lógico e Exp2.tipo == lógico) ERROR (INCOMPATIBLE_TYPES) ; senao se (Exp1.tipo != Exp2.tipo && Exp1.tipo == real ou Exp1.tipo == inteiro && Exp2.tipo = real ou Exp2.tipo == inteiro) Exp1.tipo = real senao ERROR (INCOMPATIBLE_TYPES) ; } </pre>
[17]	<pre> { se Exp5.op == and se Exp5.tipo != logico ou P1.tipo != logico ERROR (INCOMPATIBLE_TYPES) ; senao se Exp5.op == * ou Exp5.op == / se (Exp5.tipo != real e Exp5.tipo != inteiro) ou (P1.tipo != real e P1.tipo != inteiro) ERROR (INCOMPATIBLE_TYPES) ; senao se Exp5.tipo != P1.tipo Exp5.tipo = real senao se Exp5.tipo == inteiro e Exp5.op == / ERROR (INCOMPATIBLE_TYPES) ; senao se Exp5.tipo != inteiro ou Exp3.tipo != inteiro ERROR (INCOMPATIBLE_TYPES) ; senao Exp5.tipo = Exp3.tipo } </pre>
[18]	<pre> { se Exp4.tipo != logico ERROR (INCOMPATIBLE_TYPES) ; } </pre>

	}
[19]	<pre> { se Exp5.tipo == real Exp4.tipo = inteiro Exp4.end = Exp5.end senão se Exp5.tipo == inteiro Exp4.tipo = real Exp4.end = Exp5.end senão ERROR (INCOMPATIBLE_TYPES) ; } </pre>
[20]	<pre> { se isneg == ture; se Const.tipo != TOKEN_TYPE_INTEGER Const.tipo != TOKEN_TYPE_REAL ERROR (INCOMPATIBLE_TYPES) ; } </pre>

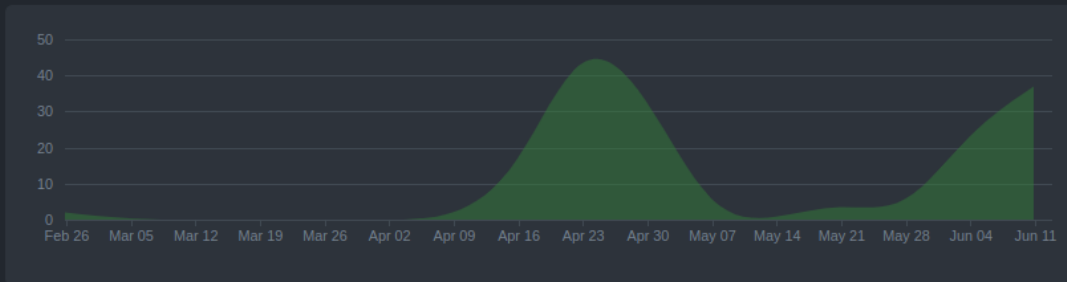
Git



Feb 26, 2023 – Jun 13, 2023

Contributions: Commits ▾

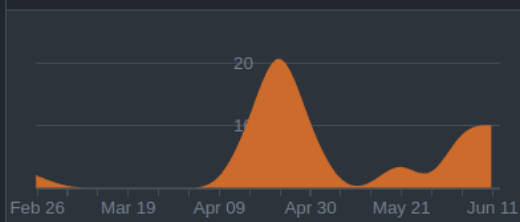
Contributions to master, excluding merge commits and bot accounts



HomeniqueM

#1

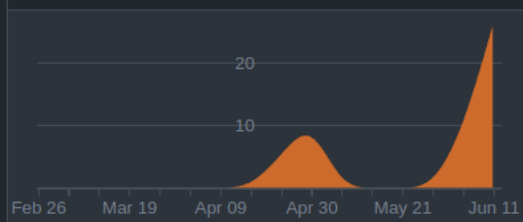
73 commits 6,164 ++ 3,708 --



IagoMorgado

#2

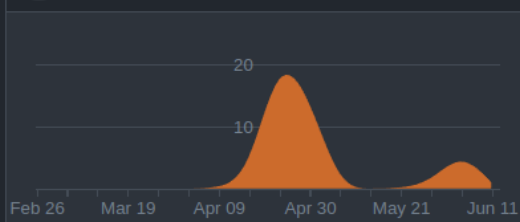
51 commits 10,609 ++ 9,047 --



Guilherme-Cosso

#3

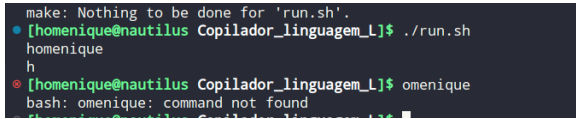
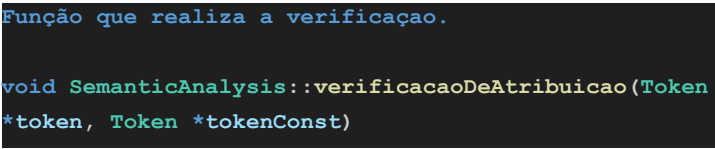
47 commits 2,026 ++ 655 --



Problemas Conhecidos

Durante o desenvolvimento do trabalho alguns erros/implementações faltantes persistiram na versão final do compilador e ainda precisam ser avaliados e corrigidos/implementados, estes são:

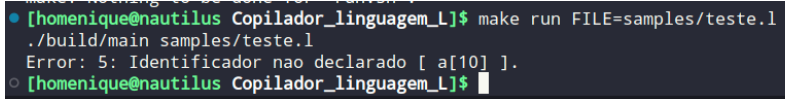
- Um vetor de caracteres não recebe uma string lida pelo teclado de forma correta salvando apenas o primeiro caractere.
- Comparações de strings não foram implementadas, porém o analisador semântico barra as operações de <>, <, >, <=, >=, permitindo apenas ==.

{ Exemplo de Erro na leitura } <code>char a[3]; readln(a); writeln(a);</code>	Saída: 
{ Exemplo de Erro na Comparação } <code>if("aaa" == "aaa") writeln("ERRO DE LEITURA");</code>	Função que realiza a verificação. 

Bônus

Posições do vetor que não foram alocadas geram erro semântico caso haja uma tentativa de acesso.

Foi usado o erro de Identificador não declarado para este caso, sendo o token lex+posição acessada.

{ Exemplo de Verificação } <code>char a[5]; a[10] = 'b';</code>	Saída: 
-----------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------