Trabalho Compiladores

Grupo: G07

Integrantes:

Gabriel Bohn - 586595
Leandro Oliveira de Queiroz - 598888
Lucas Gabriel - 591526

Análise Lexica

Autômato

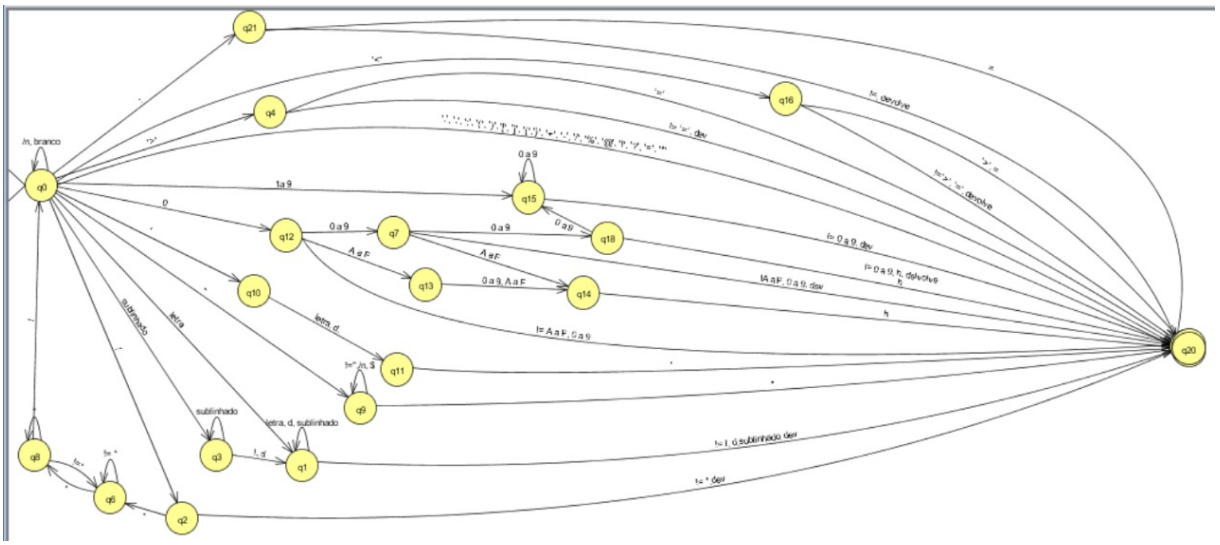

Tabela de simbolos

Tokens	Lexemas(Padrão de Formação)
id	l(l U d U _)* U _+(l U d) (l U d U _)*
const	Real, decimal, boolean, string
final	final
int	int
char	char
for	for
if	if
true	TRUE
else	else

and	and
or	or
not	not
:=	:=
=	=
(	(
)	)
<	<
>	>
<>	<>
>=	>=
<=	<=
,	,
+	+
-	-
*	*
/	/
;	;
{	{
}	}
then	then
readln	readln
false	FALSE
write	write
writeln	writeln
%	%
[	[
]	]
main	main

fim_arquivo	fim_arquivo

Ações Semânticas:

Esquema de tradução

S -> {Declaracao;} main BlocoComando fim_arquivo

Declaracao -> DeclaracaoVar | final id1 [:]= [- 36|+ 36]const 15
DeclaracaoVar -> (int | boolean | char) Indentificador {, Indentificador }

Indentificador -> id 2 12 [[:]= [- 36|+ 36]const 5 | "["const 4"]"]

BlocoComando -> "{" {Comando} "}"
Comando -> ComandosAtribuicao; | ComandosRepeticao | ComandoTeste | readln “(“ id 3 9[“[“ Exp 7 “]”] 11“)”; | write ExpEscrita; | writeln ExpEscrita; | ;
ExpEscrita ->“(“ Exp 10 {,Exp} 10 “)”

ComandosAtribuicao -> id 3 9[“[” Exp 7 7.1 ”]”] := Exp 6

ComandosRepeticao -> for “(“ [ComandosInternosRep {,ComandosInternosRep }]; Exp 8; [ComandosInternosRep {,ComandosInternosRep }]“)” (Comando | BlocoComando)

ComandosInternosRep -> ComandosAtribuicao | readln “(“ id 3 9[“[“ Exp 7“]”] 11“)” | write ExpEscrita | writeln ExpEscrita

ComandoTeste -> if “(“ Exp 8“)” then (BlocoComando | Comando) [else (BlocoComando | Comando)]

Exp -> ExpS1 53 [(= 54|<> 54|< 54|> 54|<= 54|>= 54) ExpS2 40]
ExpS -> [+ 36|- 36] T1 47 { (+ 54|- 54|or 54) T2 33 34}
T -> F1 52 { (* 54|and 54|/ 54|% 54) F2 30 31}
F -> id 3 22 ["[" Exp 25"]"] | const 24 | not F1 26 | "(" Exp 28")"

Número	Ação Semântica
1	Se id.classe != vazio ERRO: identificador ja declarado [lex].

	Se não Id.classe = “constante” Se não Id.classe = “constante”
2	Se id.classe != vazio ERRO: identificador ja declarado [lex]. Se não Id.classe = var
3	Se id.classe == vazio ERRO: identificador nao declarado [lex].
4	Se const.tipo = inteiro && 0 < const.lex && tamanho(idTipo) * const.lex <= 8kb IdTam = const.lex Se não: ERRO: tamanho do vetor excede o maximo permitido.
5	Se sinal != “” && const.tipo != inteiro || id.tipo = inteiro && (const.tipo != inteiro && const.tipo != logico) || id.tipo != inteiro && id.tipo != const.tipo ERRO: tipos incompativeis.
6	se idTamanho > 0 Se Exp2.tamanho > 0 Se idTipo == caractere Exp2.tipo == caractere || Exp2.tipo == string Se Exp2.tamanho + 1 > idTamanho ERRO: tamanho do vetor excede o maximo permitido. Se não ERRO: tipos incompativeis Se nao ERRO: tipos incompativeis Se nao ERRO: tipos incompativeis Se não Se Exp2Tamanho == 0 Se idTipo == inteiro && !(Exp2.tipo == inteiro || Exp2.tipo == logico) || idTipo == logico && !Exp2.tipo == logico || idTipo == caracter && !Exp2.tipo == caracter ERRO (tipos incompativeis)
7	Se idTam = 0 ERRO: classe de identificador incompatível [lex].
7.1	Se Exp.tipo != inteiro ERRO: expressão inválida - tipos incompativeis.
8	Se Exp.tipo != logico || Exp.tamanho != 0 ERRO: tipos incompativeis.

9	Se Id.classe == constante ERRO: classe de identificador incompatível
10	Se Exp.tamanho > 0 && Exp.tipo != string && Exp.tipo != caractere ERRO: Tipos incompativeis
11	Se id.tamanho > 0 && id.tipo != caractere ERRO: classe de identificador incompatível [lex].
12	idTipo = reglex.tipo
15	Se const.Tamanho() > 0 ERRO tipos incompativeis Se sinal != “” && const.tipo != inteiro ERRO identificador ja declarado Se não Id.tipo = const.tipo;
22	FTipo = id.tipo FTamanho = id.tamanho
24	FTipo = const.tipo FTamanho = const.tamanho
25	Se idTamanho == 0 ERRO Se ExpTipo != inteiro ERRO FTamanho = 0
26	Se F1.tipo == logico F1.Tamanho == 0 FTipo = F1.tipo; FTamanho = F1.tamanho; Se não ERRO: tipos incompativeis.
28	FTipo = Exp.tipo ; FTamanho = Exp.tamanho;
30	Se op == “*” && (F1Tipo != inteiro || F2.tipo != inteiro) || op == “and” && (F1Tipo != logico || F2.tipo != logico) || op == “/” && (F1Tipo != inteiro || F2.tipo != inteiro) || op == “%” && (F1Tipo != inteiro || F2.tipo != inteiro) || F1Tipo != F2.tipo ERRO (tipos incompativeis) TTipo = F1.tipo TTamanho = F1.tamanho

31	Se F2Tamanho > 0 || F1Tamanho > 0 ERRO:classe de identificador incompativel [lex].
33	Se op == “+” && (T1Tipo != inteiro || T2.tipo != inteiro) || op == “-” && (T1Tipo != inteiro || T2.tipo != inteiro) || op == “or” && (T1Tipo != logico || T2.tipo != logico) || T1Tipo != T2.tipo ERRO (tipos incompatives)
34	Se T2.tamanho > 0 || T1Tamanho > 0 ERRO:classe de identificador incompativel [lex].
36	sinal = reglex.lexema
40	Se op == “=” Se ExpS1Tipo != “string” || ExpS1Tipo != “caractere”” && ExpS2Tipo != “string” || ExpS2Tipo != “caractere” Se ExpS1Tipo != ExpS2Tipo ERRO: tipos incompativeis Senao Se ExpS1Tamanho > 0 || ExpS2Tamanho > 0 Erro tipos incompativeis Senao Se ExpS1Tipo != ExpS2Tipo ERRO: tipos incompativeis
47	Se sinal != “” && T1.tipo != inteiro ERRO ExpSTipo = T1.tipo ExpSTamanho = T1.tamanho
52	TTipo = F1.tipo TTamanho = F1.tamanho
53	ExpTipo = ExpS1.tipo ExpTamanho = ExpS1.tamanho
54	op = reglex.lexema

Geração de Código

Esquema de tradução

S -> {Declaracao;} 2 main BlocoComando fim_arquivo 3

Declaracao -> DeclaracaoVar | final id [:]= [- |+]const 1
DeclaracaoVar -> (int | boolean | char) Indentificador {, Indentificador }

Indentificador -> id [[:]= [- |+]const | "["const"]"] 1

BlocoComando -> "{" {Comando} "}"
Comando -> ComandosAtribuicao; | ComandosRepeticao | ComandoTeste | readln “(“ id [“[“ Exp “]”] “)”; | write ExpEscrita; | writeln ExpEscrita; | ;
ExpEscrita ->“(“ Exp {,Exp} “)”

ComandosAtribuicao -> id[“[” Exp ”]”] := Exp 4

ComandosRepeticao -> for “(“ [ComandosInternosRep {,ComandosInternosRep }]; 5.1 Exp; [ComandosInternosRep 5.2 {,ComandosInternosRep }]“)” (Comando | BlocoComando) 5.3

ComandosInternosRep -> ComandosAtribuicao | readln “(“ id[“[“ Exp “]”]“)” | write ExpEscrita | writeln ExpEscrita

ComandoTeste -> if 6.1 “(“ Exp 6.2“)” then (BlocoComando | Comando) 6.3[else (BlocoComando | Comando)] 6.6

Exp -> ExpS1 7 [(=|<>|<|>|<=|>=) ExpS2 8]
ExpS -> [+|-] T1 9 { (+|-|or) T2 10}
T -> F1 11 { (*|and|/|%) F2 12}
F -> id 13 ["[" Exp"]" 14] | const 15 | not F1 16 | "(" Exp")" 17

Número	Regra de Geração de Código (GC)
1	Se idTipo == inteiro Se idTamanho > 0 sword idTamanho DUP (?) Se não Se exp sword exp.lexema Se nao sword ? Se não Se idTipo == caractere Se idTamanho > 0 Se exp dseg SEGMENT PUBLIC byte exp.lexema$ dseg ENDS Se nao Se não Se exp byte exp.lexema Se nao byte ? Se não Se idTipo == logico Se idTamanho > 0 byte idTamanho DUP (?) Se não Se exp

	byte exp.lexema Se nao byte ?
2	dseg ENDS ; fim seg. dados cseg SEGMENT PUBLIC ; inicio seg. de codigo ASSUME CS:cseg, DS:dseg strt:
3	mov ah, 4Ch int 21h ; parar execucao do programa cseg ENDS ; fim seg. de codigo END strt ; fim do programa

4	Se Exp2.tamanho > 0
	Para i no tamanho de Exp2
	mov ax,DS:[Exp2.endereco+i]
	mov DS:[id.endereco], ax
	Se nao
	mov ax,DS:[Exp2.endereco]
	mov DS:[id.endereco], ax

5.1	rotInicio = novoRot();
	rotFim = novoRot();
	rotInicio:

5.2	mov ax, DS:[Exp.endereco]
	cmp ax,0
	je rotFim

5.3	jmp rotInicio rotFim:
6.1	rotFalso = novoRot(); rotFim = novoRot();

6.2	mov ax, DS:[Exp.endereco]
	cmp ax, 0
	je rotFalso

6.3	jmp rotFim rotFalso:
6.4	RotFim:
7	Exp.endereco = S1.endereco

8	mov ax, DS:[Exp.endereco]
	mov bx, DS:[S1.endereco]
	
	cmp ax, bx
	rotVerdadeiro = novoRot();
	
	Se op == "=" {

	je rotVerdadeiro
	} Se nao se op == "<>" {
	jne rotVerdadeiro
	} Se nao se op == "<" {
	jl rotVerdadeiro
	} Se nao se op == ">" {
	jg rotVerdadeiro
	} Se nao se op == ">=" {
	jge rotVerdadeiro
	} Se nao se op == "<=" {
	jle rotVerdadeiro
	}
	
	mov ax, 0
	
	rotFim = novoRot();
	jmp rotFim
	rotVerdadeiro:
	
	mov ax, 1
	rotFim:
	
	Exp.endereco = novoTemp()
	mov Exp.endereco, DS:[ax]

9	ExpS.endereco = T1.endereco
	
	mov ax, DS:[T1.endereco]
	neg ax
	add ax 1
	mov DS:[ExpS.endereco], ax

10	mov ax, DS:[ExpS.endereco]
	mov bx, DS:[T2.endereco]
	
	Se op == "+" {
	add ax bx
	} Se nao se op == "-" {
	neg bx
	add ax bx
	} Se nao se op == "or" {
	imul bx
	neg ax
	add ax 1
	}
	ExpS.endereco = novoTemp()

	mov ExpS.endereco, DS:[ax]
11	T.endereco = F1.endereco

12	mov ax, DS:[T.endereco]
	mov bx, DS:[F2.endereco]
	
	Se op == "*" {
	imul bx
	} Se nao se op == "/" {
	idiv bx
	} Se nao se op == "and" {
	imul bx
	} Se nao se op == "%" {
	idiv bx
	mov ax dx
	}
	T.endereco = novoTemp()
	mov T.endereco, DS:[ax]

13	F.endereco = id.endereco

14	F.endereco = novoTemp()
	mov ax, DS:[Exp.endereco]
	
	Se Exp.tipo == inteiro
	add ax, DS:[Exp.endereco]
	add ax, DS:[id.endereco]
	mov F.endereco, DS:[ax]

15	Se const.tipo == string
	dseg SEGMENT PUBLIC
	byte regConst.getLexema()$
	dseg ENDS
	F.endereco = novoTemp()
	Se nao
	F.endereco = novoTemp()
	mov ax, const.lexema
	mov DS:[F.endereco], ax

16	mov ax, DS:[F1.endereco]
	neg ax
	add ax 1
	mov DS:[F.endereco], ax

17	F.endereco = Exp.endereco