



CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

DEPARTAMENTO DE COMPUTAÇÃO

COMPILADORES

TRABALHO PRÁTICO - IMPLEMENTAÇÃO DE UM COMPILADOR

Professora: Kecia Marques

Aluno:

Francisco Abreu Gonçalves

1. Introdução

O trabalho prático realizado na disciplina de Compiladores é a construção de um compilador completo para uma dada linguagem de programação. Esse trabalho foi dividido em etapas com base na estrutura de um compilador, o qual possui uma parte de **Análise** e **Síntese**. Dessa forma, a subdivisão do projeto se deu da seguinte forma:

a) ANÁLISE :

- 1) Analisador Léxico e Tabela de símbolos
- 2) Analisador Sintático
- 3) Analisador Semântico

b) SÍNTESE:

- 1) Gerador de Código

A implementação do compilador foi feita na linguagem C++, no editor de texto Visual Studio e utilizou-se a plataforma GitHub para o controle de versão do código.

O código fonte do compilador pode ser consultado por meio do repositório: https://github.com/Francis1408/Generic_Compiler

2. Análise

Na parte da análise, ou *front-end*, do compilador, recebe-se como entrada o código fonte da linguagem a ser compilada. A partir disso, cabe à Análise de verificar se o programa está escrito de forma correta, seguindo a gramática da linguagem e o padrão de formação dos tokens.

Como forma de armazenar valores de variáveis e retorno de funções, o compilador, nesta parte, conta com a ajuda da **Tabela de símbolos**. Cada tipo de análise feita no *front-end* é descrita a seguir.

2.1 Análise Léxica

Na Análise léxica, o compilador lê o arquivo do programa fonte, caractere por caractere e os agrupa em sequências significativas denominadas **Lexemas**. Na implementação, o Lexema é uma estrutura

(*struct*) que possui dois parâmetros: **token** do tipo *string* e um **tipo de token** do tipo *TokenType*:

```
struct Lexeme {
    std::string token;
    enum TokenType type;

    Lexeme() : token(""), type(TT_END_OF_FILE) {}
    virtual ~Lexeme() {}

    const std::string str() {
        std::stringstream ss;

        ss << "(" << token << "\", " << tt2str(type) << ")";
        return ss.str();
    }
};
```

Imagem 1: Estrutura de um Lexema

O *TokenType*, por sua vez, é uma enumeração de tokens presentes na gramática da linguagem, indo de -1 até 37. Essa lista é mostrada abaixo:

```
enum TokenType{
    // Specials
    TT_UNEXPECTED_EOF = -2,
    TT_INVALID_TOKEN = -1,
    TT_END_OF_FILE = 0,

    //Symbols
    TT_SEMICOLON, // ;
    TT_COMMA, // ,
    TT_PERIOD, // .
    TT_ASSIGN, // =
```

```
TT_POINTS, // :
TT_PAR1, // (
TT_PAR2, // )
TT_CHAV1, // {
TT_CHAV2, // }
TT_QUOTE, // "

//Logic Operators
TT_EQUAL, // ==
TT_NOT_EQUAL, // !=
TT_LOWER, // <
TT_GREATER, // >
TT_LESS_EQUAL, // <=
TT_GREATER_EQUAL, // >=

// Conector Operators
TT_AND, // &&
TT_OR, // ||
TT_NOT, // !

// Arithmetic operators
TT_ADD, // +
TT_SUB, // -
TT_MUL, // *
TT_DIV, // /

//Keywords
TT_CLASS, // class
TT_IF, // if
TT_ELSE, // else
TT_WHILE, // while
TT_WRITE, // write
```

```
TT_READ, // read
TT_DO, // do
TT_INT, // int
TT_FLOAT, // float
TT_STRING, // string

//Others
TT_ID, // variable
TT_INTEGER, // Integer
TT_REAL, // Real number
TT_LITERAL, // String
};
```

Imagem 1: Enumeração dos tokens

Para a identificação do padrão desses tokens, desenvolveu-se por meio da gramática da linguagem um Autômato Finito Determinismo (AFD):

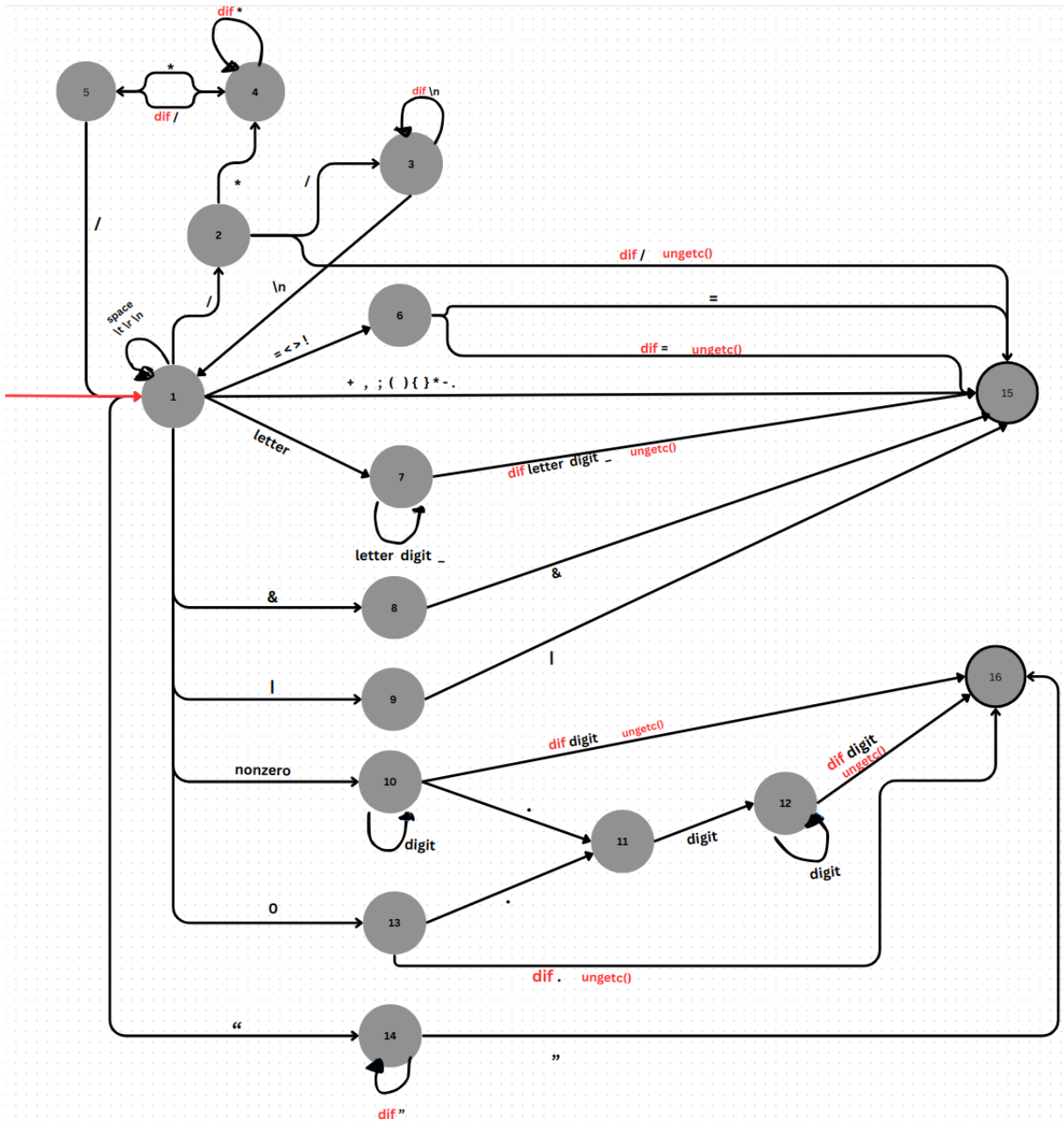


Imagem 3: AFD de formação dos tokens

Logo, o funcionamento da análise léxica do código se dá por meio da dinâmica a seguir:

1. No arquivo principal (compler.cpp), inicia-se a execução lendo o arquivo fonte, caractere por caractere, o qual é salvo na variável *char c*. Enquanto não chegar ao final do arquivo (TT_END_OF_FILE = 0), a classe *LexicalAnalysis* invoca a função *nextToken()*, a qual consome o próximo caractere do arquivo e o concatena na *string token*.

2. A partir do caractere consumido, a função *nextToken()* avança na AFD. Caso chegue em um estado final (15, 16), um lexema foi encontrado.

Assim, o programa recupera o último valor lido (*ungetc()*) e envia-o para ser encontrado na **tabela de símbolos**.

3. O método *find()* da classe SymbolTable realiza a procura do lexema formado. Caso não encontre-o, ele é salvo como um TT_IDENTIFIER.

Obs: Esse método é invocado apenas ao chegar ao estado final 15. No caso do estado 16, já se conhece o tipo do token.

4. Se os caracteres consumidos não levarem a nenhum estado do AFD, ou o arquivo fonte termina antes de se chegar à um estado final, o programa denuncia um TT_INVALID_TOKEN e TT_UNEXPECTED_EOF, respectivamente. Assim, o programa é abortado.

5. Ao final do processo, a sequência de tokens identificados é impressa, assim como a **tabela de símbolos**.

TESTES:

Teste1:

```
class Teste1
int a,b,c;
float result;
{
    write("Digite o valor de a:");
    read (a);
    write("Digite o valor de c:");
    read (c);
    b = 10;
    result = (a * c)/(b 5 - 345);
    write("O resultado e: ");
    write(result);
}
```

```
("class", CLASS)
("Teste1", IDENTIFIER)
("int", INT)
("a", IDENTIFIER)
(",", COMMA)
("b", IDENTIFIER)
(",", COMMA)
("c", IDENTIFIER)
(";", SEMICOLON)
("float", FLOAT)
("result", IDENTIFIER)
(";", SEMICOLON)
("{", CHAV1)
("write", WRITE)
("(", PAR1)
("Digite o valor de a:", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("a", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite o valor de c:", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("c", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("b", IDENTIFIER)
("=", ASSIGN)
("10", INTEGER)
(";", SEMICOLON)
("result", IDENTIFIER)
("=", ASSIGN)
("(", PAR1)
("a", IDENTIFIER)
("*", MUL)
("c", IDENTIFIER)
(")", PAR2)
("/", DIV)
("(", PAR1)
("b", IDENTIFIER)
("5", INTEGER)
("-", SUB)
("345", INTEGER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("O resultado e: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("result", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("}", CHAV2)
("", END_OF_FILE)
```


-----TABELA DE SÍMBOLOS-----	
LEXEMA	TOKEN ID
!	19
!=	12
&&	17
(6
)	7
*	22
+	20
,	2
-	21
.	3
/	23
:	5
;	1
<	13
<=	15
=	4
==	11
>	14
>=	16
Teste1	34
a	34
b	34
c	34
class	24
do	30
else	26
float	32
if	25
int	31
read	29
result	34
string	33
while	27
write	28
{	8
	18
}	9

Teste2:

class Teste2

/* Teste de comentário

com mais de uma linha

a, 9valor, b_1, b_2 : int;

```

{
write("Entre com o valor de a: ");
read (a);
b_1 := a * a;
write("O valor de b1 e: ");
write (b_1);
b_2 = b + a/2 * (a + 5);
write("O valor de b2 e: ");
Write (b2);
}

```

```

("class", CLASS)
("Teste2", IDENTIFIER)
("", END_OF_FILE)

```

-----TABELA DE SÍMBOLOS-----	
LEXEMA	TOKEN ID
!	19
!=	12
&&	17
(6
)	7
*	22
+	20
,	2
-	21
.	3
/	23
:	5
;	1
<	13
<=	15
=	4
==	11
>	14
>=	16
Teste2	34
class	24
do	30
else	26
float	32
if	25
int	31
read	29
string	33
while	27
write	28
{	8
	18
}	9

Teste2 (CORRIGIDO):

```
class Teste2
/* Teste de comentário
com mais de uma linha */
a, 9valor, b_1, b_2, int;
{
write("Entre com o valor de a: ");
read (a);
b_1 = a * a;
write("O valor de b1 e: ");
write (b_1);
b_2 = b + a/2 * (a + 5);
write("O valor de b2 e: ");
Write (b2);
}
```

```
("class", CLASS)
("Teste2", IDENTIFIER)
("a", IDENTIFIER)
(",", COMMA)
("9", INTEGER)
("valor", IDENTIFIER)
(",", COMMA)
("b_1", IDENTIFIER)
(",", COMMA)
("b_2", IDENTIFIER)
(",", COMMA)
("int", INT)
(";", SEMICOLON)
("{", CHAV1)
("write", WRITE)
("(", PAR1)
("""Entre com o valor de a: """, LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("a", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("b_1", IDENTIFIER)
("=", ASSIGN)
("a", IDENTIFIER)
("*", MUL)
("a", IDENTIFIER)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("""O valor de b1 e: """, LITERAL)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("b_1", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("b_2", IDENTIFIER)
("=", ASSIGN)
("b", IDENTIFIER)
("+", ADD)
("a", IDENTIFIER)
("/", DIV)
("2", INTEGER)
("*", MUL)
("(", PAR1)
("a", IDENTIFIER)
("+", ADD)
("5", INTEGER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("""O valor de b2 e: """, LITERAL)
(")", PAR2)
(";", SEMICOLON)
("Write", IDENTIFIER)
("(", PAR1)
("b2", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("}", CHAV2)
("", END_OF_FILE)
```

-----TABELA DE SÍMBOLOS-----		
LEXEMA	TOKEN	ID
!		19
!=		12
&&		17
(6
)		7
*		22
+		20
,		2
-		21
.		3
/		23
:		5
;		1
<		13
<=		15
=		4
==		11
>		14
>=		16
Teste2		34
Write		34
a		34
b		34
b2		34
b_1		34
b_2		34
class		24
do		30
else		26
float		32
if		25
int		31
read		29
string		33
valor		34
while		27
write		28
{		8
		18
}		9

Teste3:

classe Teste3

/** Verificando fluxo de controle

```
Programa com if e while aninhados **/  
int i;  
int media, soma;  
{  
soma = 0;  
write("Quantos dados deseja informar?" );  
read (qtd);  
IF (qtd>=2){  
i=0;  
do{  
write("Altura: ");  
read (altura);  
soma = soma+altura;  
i = i + 1;  
}while( i < qtd);  
media = soma / qtd;  
write("Media: ");  
write (media);}  
else{  
write("Quantidade inválida.");  
}  
}
```

```
("classe", IDENTIFIER)
("Teste3", IDENTIFIER)
("int", INT)
("i", IDENTIFIER)
(";", SEMICOLON)
("int", INT)
("media", IDENTIFIER)
(",", COMMA)
("soma", IDENTIFIER)
(";", SEMICOLON)
("{", CHAV1)
("soma", IDENTIFIER)
("=", ASSIGN)
("0", INTEGER)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Quantos dados deseja informar?", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("qtd", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("IF", IDENTIFIER)
("(", PAR1)
("qtd", IDENTIFIER)
(">=", GREATER_EQUAL)
("2", INTEGER)
(")", PAR2)
("{", CHAV1)
("i", IDENTIFIER)
("=", ASSIGN)
("0", INTEGER)
(";", SEMICOLON)
("do", DO)
("{", CHAV1)
("write", WRITE)
("(", PAR1)
("Altura: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("altura", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("soma", IDENTIFIER)
("=", ASSIGN)
("soma", IDENTIFIER)
("+", ADD)
("altura", IDENTIFIER)
(";", SEMICOLON)
```

```
(  
    "i", IDENTIFIER)  
    ("=", ASSIGN)  
    ("i", IDENTIFIER)  
    ("+", ADD)  
    ("1", INTEGER)  
    (";", SEMICOLON)  
    ("}", CHAV2)  
    ("while", WHILE)  
    ("(", PAR1)  
    ("i", IDENTIFIER)  
    ("<", LOWER)  
    ("qtd", IDENTIFIER)  
    (")", PAR2)  
    (";", SEMICOLON)  
    ("media", IDENTIFIER)  
    ("=", ASSIGN)  
    ("soma", IDENTIFIER)  
    ("/", DIV)  
    ("qtd", IDENTIFIER)  
    (";", SEMICOLON)  
    ("write", WRITE)  
    ("(", PAR1)  
    ("Media: ", LITERAL)  
    (")", PAR2)  
    (";", SEMICOLON)  
    ("write", WRITE)  
    ("(", PAR1)  
    ("media", IDENTIFIER)  
    (")", PAR2)  
    (";", SEMICOLON)  
    ("}", CHAV2)  
    ("else", ELSE)  
    ("{" , CHAV1)  
    ("write", WRITE)  
    ("(", PAR1)  
    ("Quantidade inválida.", LITERAL)  
    (")", PAR2)  
    (";", SEMICOLON)  
    ("}", CHAV2)  
    ("}", CHAV2)  
    ("", END_OF_FILE)
```


-----TABELA DE SÍMBOLOS-----

LEXEMA	TOKEN ID
!	19
!=	12
&&	17
(6
)	7
*	22
+	20
,	2
-	21
.	3
/	23
:	5
;	1
<	13
<=	15
=	4
==	11
>	14
>=	16
IF	34
Teste3	34
altura	34
class	24
classe	34
do	30
else	26
float	32
i	34
if	25
int	31
media	34
qtd	34
read	29
soma	34
string	33
while	27
write	28
{	8
	18
}	9

Teste4:

```
{  
// Outro programa de teste  
int idade, j, k, @total;  
string nome, texto;  
write("Digite o seu nome: ");  
read(nome);  
write("Digite o seu sobrenome");  
read(sobrenome);  
write("Digite a sua idade: ");  
read (idade);  
k := i * (5-i * 50 / 10;  
j := i * 10;  
k := i * j / k;  
texto = nome + " " + sobrenome + ", os números gerados sao: ";  
write (text);  
write(j);  
write(k);  
}
```

```
("{" , CHAV1)
("int" , INT)
("idade" , IDENTIFIER)
("," , COMMA)
("j" , IDENTIFIER)
("," , COMMA)
("k" , IDENTIFIER)
("," , COMMA)
("@" , INVALID_TOKEN)
```

-----TABELA DE SÍMBOLOS-----

LEXEMA	TOKEN ID
!	19
!=	12
&&	17
(6
)	7
*	22
+	20
,	2
-	21
.	3
/	23
:	5
;	1
<	13
<=	15
=	4
==	11
>	14
>=	16
class	24
do	30
else	26
float	32
idade	34
if	25
int	31
j	34
k	34
read	29
string	33
while	27
write	28
{	8
	18
}	9

Teste4(Corrigido):

```
{  
// Outro programa de teste  
int idade, j, k, total;  
string nome, texto;  
write("Digite o seu nome: ");  
read(nome);  
write("Digite o seu sobrenome");  
read(sobrenome);  
write("Digite a sua idade: ");  
read (idade);  
k = i * (5-i * 50 / 10);  
j = i * 10;  
k = i * j / k;  
texto = nome + " " + sobrenome + ", os números gerados sao: ";  
write (text);  
write(j);  
write(k);  
}
```

```
("{" , CHAV1)
("int" , INT)
("idade" , IDENTIFIER)
("," , COMMA)
("j" , IDENTIFIER)
("," , COMMA)
("k" , IDENTIFIER)
("," , COMMA)
("total" , IDENTIFIER)
("; " , SEMICOLON)
("string" , STRING)
("nome" , IDENTIFIER)
("," , COMMA)
("texto" , IDENTIFIER)
("; " , SEMICOLON)
("write" , WRITE)
("(" , PAR1)
("Digite o seu nome:" , LITERAL)
(")" , PAR2)
("; " , SEMICOLON)
("read" , READ)
("(" , PAR1)
("nome" , IDENTIFIER)
(")" , PAR2)
("; " , SEMICOLON)
("write" , WRITE)
("(" , PAR1)
("Digite o seu sobrenome" , LITERAL)
(")" , PAR2)
("; " , SEMICOLON)
("read" , READ)
("(" , PAR1)
("sobrenome" , IDENTIFIER)
(")" , PAR2)
("; " , SEMICOLON)
("write" , WRITE)
("(" , PAR1)
("Digite a sua idade: " , LITERAL)
(")" , PAR2)
("; " , SEMICOLON)
("read" , READ)
("(" , PAR1)
("idade" , IDENTIFIER)
(")" , PAR2)
("; " , SEMICOLON)
("k" , IDENTIFIER)
("=" , ASSIGN)
("i" , IDENTIFIER)
("*" , MUL)
("(" , PAR1)
("5" , INTEGER)
("-" , SUB)
("i" , IDENTIFIER)
("*" , MUL)
("50" , INTEGER)
```

```
("/", DIV)
("10", INTEGER)
(";", SEMICOLON)
("j", IDENTIFIER)
("=", ASSIGN)
("i", IDENTIFIER)
("*", MUL)
("10", INTEGER)
(";", SEMICOLON)
("k", IDENTIFIER)
("=", ASSIGN)
("i", IDENTIFIER)
("*", MUL)
("j", IDENTIFIER)
("/", DIV)
("k", IDENTIFIER)
(";", SEMICOLON)
("texto", IDENTIFIER)
("=", ASSIGN)
("nome", IDENTIFIER)
("+", ADD)
(" " " ", LITERAL)
("+", ADD)
("sobrenome", IDENTIFIER)
("+", ADD)
(" ", os números gerados sao: " ", LITERAL)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("text", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("j", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("k", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("}", CHAV2)
(" ", END_OF_FILE)
```

-----TABELA DE SÍMBOLOS-----		
LEXEMA	TOKEN	ID
!	19	
!=	12	
&&	17	
(6	
)	7	
*	22	
+	20	
,	2	
-	21	
.	3	
/	23	
:	5	
;	1	
<	13	
<=	15	
=	4	
==	11	
>	14	
>=	16	
class	24	
do	30	
else	26	
float	32	
i	34	
idade	34	
if	25	
int	31	
j	34	
k	34	
nome	34	
read	29	
sobrenome	34	
string	33	
text	34	
texto	34	
total	34	
while	27	
write	28	
{	8	
	18	
}	9	

Teste5:

```
class MinhaClasse
{
float a, b, c;
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior := 0;
if ( a>b && a>c )
maior = a;
else
if (b>c)
maior = b;
else
maior = c;
write("O maior número é: ");
write(maior);
```



```
("class", CLASS)
("MinhaClasse", IDENTIFIER)
("{", CHAV1)
("float", FLOAT)
("a", IDENTIFIER)
(",", COMMA)
("b", IDENTIFIER)
(",", COMMA)
("c", IDENTIFIER)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite um número", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("a", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite outro número: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("b", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite mais um número: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("c", IDENTIFIER)
(";", SEMICOLON)
("maior", IDENTIFIER)
(":", INVALID_TOKEN)
```

-----TABELA DE SÍMBOLOS-----

LEXEMA	TOKEN ID
!	19
!=	12
&&	17
(6
)	7
*	22
+	20
,	2
-	21
.	3
/	23
:	5
;	1
<	13
<=	15
=	4
==	11
>	14
>=	16
MinhaClasse	34
a	34
b	34
c	34
class	24
do	30
else	26
float	32
if	25
int	31
maior	34
read	29
string	33
while	27
write	28
{	8
	18
}	9

Teste5 (Corrigido):

```
class MinhaClasse
```

```
{
```

```
float a, b, c;
```

```
write("Digite um número");
```

```
read(a);
```

```
write("Digite outro número: ");
```

```
read(b);
```

```
write("Digite mais um número: ");
```

```
read(c;
```

```
maior = 0;
```

```
if ( a>b && a>c )
```

```
maior = a;
```

```
else
```

```
if (b>c)
```

```
maior = b;
```

```
else
```

```
maior = c;
```

```
write("O maior número é: ");
```

```
write(maior);
```

```
("class", CLASS)
("MinhaClasse", IDENTIFIER)
("{", CHAV1)
("float", FLOAT)
("a", IDENTIFIER)
(",", COMMA)
("b", IDENTIFIER)
(",", COMMA)
("c", IDENTIFIER)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite um número", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("a", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite outro número: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("b", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("Digite mais um número: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("read", READ)
("(", PAR1)
("c", IDENTIFIER)
(";", SEMICOLON)
("maior", IDENTIFIER)
("=", ASSIGN)
("0", INTEGER)
(";", SEMICOLON)
("if", IF)
("(", PAR1)
("a", IDENTIFIER)
(">", GREATER)
("b", IDENTIFIER)
("&&", AND)
("a", IDENTIFIER)
(">", GREATER)
("c", IDENTIFIER)
```

```
(")", PAR2)
("maior", IDENTIFIER)
("=", ASSIGN)
("a", IDENTIFIER)
(";", SEMICOLON)
("else", ELSE)
("if", IF)
("(", PAR1)
("b", IDENTIFIER)
(">", GREATER)
("c", IDENTIFIER)
(")", PAR2)
("maior", IDENTIFIER)
("=", ASSIGN)
("b", IDENTIFIER)
(";", SEMICOLON)
("else", ELSE)
("maior", IDENTIFIER)
("=", ASSIGN)
("c", IDENTIFIER)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("O maior número é: ", LITERAL)
(")", PAR2)
(";", SEMICOLON)
("write", WRITE)
("(", PAR1)
("maior", IDENTIFIER)
(")", PAR2)
(";", SEMICOLON)
("", END_OF_FILE)
```

-----TABELA DE SÍMBOLOS-----

LEXEMA	TOKEN ID
!	19
!=	12
&&	17
(6
)	7
*	22
+	20
,	2
-	21
.	3
/	23
:	5
;	1
<	13
<=	15
=	4
==	11
>	14
>=	16
MinhaClasse	34
a	34
b	34
c	34
class	24
do	30
else	26
float	32
if	25
int	31
maior	34
read	29
string	33
while	27
write	28
{	8
	18
}	9

2.1 Análise Sintática

Na Análise Sintática, o compilador consome os lexemas gerados pelo analisador sintático e monta, assim, uma árvore de derivação. Para isso, é necessário que ele consulte a gramática da linguagem, de forma a verificar se os lexemas encontrados estão na sequência correta.

Na implementação, foi utilizado a seguinte gramática LL(1):

<code><program></code>	<code>::= class identifier [<decl-list>] <body></code>
<code><decl-list></code>	<code>::= <decl> “,” {<decl>, “;”}</code>
<code><decl></code>	<code>::= <type> <ident-list></code>
<code><ident-list></code>	<code>::= identifier {“,” identifier}</code>
<code><type></code>	<code>::= int string float</code>
<code><body></code>	<code>::= “{” <stmt-list> “}”</code>
<code><stmt-list></code>	<code>::= <stmt> “;” {<stmt> “;”}</code>
<code><stmt></code>	<code>::= <assign-stmt> <if-stmt> <do-stmt> <read-stmt> </code>
<code><write-stmt></code>	
<code><assign-stmt></code>	<code>::= identifier “=” <simple_expr></code>
<code><if-stmt></code>	<code>::= if “(” <condition> “)” “{” <stmt-list> “}” <if-stmt’></code>
<code><if-stmt’></code>	<code>::= λ else “{” <stmt-list> “}”</code>
<code><condition></code>	<code>::= <expression></code>
<code><do-stmt></code>	<code>::= do “{” <stmt-list> “}” <do-suffix></code>
<code><do-suffix></code>	<code>::= while “(” <condition> “)”</code>
<code><read-stmt></code>	<code>::= read “(” identifier “)”</code>
<code><write-stmt></code>	<code>::= write “(” <writable> “)”</code>
<code><writable></code>	<code>::= <simple-expr></code>
<code><expression></code>	<code>::= <simple-expr> <expression’></code>
<code><expression’></code>	<code>::= λ <relop> <simple-expr></code>
<code><simple-expr></code>	<code>::= <term> <simple-expr’></code>
<code><simple-expr’></code>	<code>::= <addop> <term> <simple-expr’> λ</code>
<code><term></code>	<code>::= <factor-a> <term’></code>
<code><term’></code>	<code>::= <mulop> <factor-a> <term’> λ</code>
<code><factor-a></code>	<code>::= <factor> “!” <factor> “-” <factor></code>
<code><factor></code>	<code>::= identifier <constant> “(” <expression> “)”</code>
<code><relop></code>	<code>::= “>” “>=” “<” “<=” “!=” “==”</code>

`<addop>` ::= "+" | "-" | "||"
`<mulop>` ::= "*" | "/" | "&&"

`<constant>` ::= *integer_const* | *literal* | *real_const*

No código, o analisador sintático é uma classe cuja cada símbolo não terminal da gramática é visto como um método. Dessa forma, seguindo a derivação a direita, cada método pode chamar outros métodos ligados à símbolos não terminais para e encontrar uma sequência válida.

Caso a regra encontre um símbolo terminal, o método irá chamar o método **eat()** para consumir o token esperado. Abaixo, tem-se o exemplo da implementação do método relacionado ao símbolo não terminal `<body>` e do método **eat()**.

```
// <body> ::= "{" <stmt-list> "}"
void SyntaticAnalysis::procBody() {
    eat(TT_CHAV1);
    procStmt_list();
    eat(TT_CHAV2);
}
```

Imagem 4: Método `<body>`

```
void SyntaticAnalysis::eat(enum TokenType type) {

    if(type == m_current.type && type != TT_END_OF_FILE) {
        advance();
    } else if (type == TT_END_OF_FILE) {
        std::cout << "Análise léxica feita com sucesso!" <<
        std::endl;
        advance();
    } else {
        showError();
    }
}
```



```
}
```

Imagem 4: Método eat()

No método eat(), o token esperado na sequência é passado como parâmetro e comparado com o token atual. Caso sejam iguais, a análise continua. Caso contrário, a análise será interrompida, mostrando o número da linha que ocorreu o erro e o seu tipo. Se o arquivo chegar ao seu final sem nenhum problema a priori, o código está sintaticamente correto e a análise finalizará.

Os exemplos testes são os mesmos da análise léxica, porém sem nenhum erro léxico:

TESTES:

Teste1:

```
class Teste1
int a,b,c;
float result;
{
    write("Digite o valor de a:");
    read (a);
    write("Digite o valor de c:");
    read (c);
    b = 10;
    result = (a * c)/(b5 - 345);
    write("O resultado e: ");
    write(result);
}
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste1.txt
Erro na linha: 10
Lexema não esperado [5]
```

Teste1:

```
class Teste1
int a,b,c;
float result;
```

```

{
    write("Digite o valor de a:");
    read (a);
    write("Digite o valor de c:");
    read (c);
    b = 10;
    result = (a * c)/(b*5 - 345);
    write("O resultado e: ");
    write(result);
}

```

```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste1.txt
Análise sintática feita com sucesso!

```

Teste2:

```

class Teste2
/* Teste de comentário
com mais de uma linha */
a, 9valor, b_1, b_2, int;
{
    write("Entre com o valor de a: ");
    read (a);
    b_1 = a * a;
    write("O valor de b1 e: ");
    write (b_1);
    b_2 = b + a/2 * (a + 5);
    write("O valor de b2 e: ");
    Write (b2);
}

```

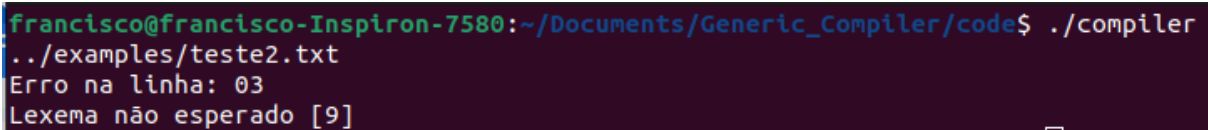
```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste2.txt
Erro na linha: 03
Lexema não esperado [a]

```

Teste2:

```
class Teste2
/* Teste de comentário
com mais de uma linha */
int a, 9valor, b_1, b_2, int;
{
write("Entre com o valor de a: ");
read (a);
b_1 = a * a;
write("O valor de b1 e: ");
write (b_1);
b_2 = b + a/2 * (a + 5);
write("O valor de b2 e: ");
Write (b2);
}
```

A terminal window with a dark background. The prompt is 'francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code\$'. The user has entered './compiler ../examples/teste2.txt'. The output shows an error: 'Erro na linha: 03' and 'Lexema não esperado [9]'.

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste2.txt
Erro na linha: 03
Lexema não esperado [9]
```

Teste2:

```
class Teste2
/* Teste de comentário
com mais de uma linha */
int a, valor, b_1, b_2, int;
{
write("Entre com o valor de a: ");
read (a);
b_1 = a * a;
write("O valor de b1 e: ");
write (b_1);
b_2 = b + a/2 * (a + 5);
write("O valor de b2 e: ");
Write (b2);
}
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste2.txt
[Erro na linha: 03
Lexema não esperado [int]
```

Teste2:

```
class Teste2
/* Teste de comentário
com mais de uma linha */
int a, valor, b_1, b_2;
{
write("Entre com o valor de a: ");
read (a);
b_1 = a * a;
write("O valor de b1 e: ");
write (b_1);
b_2 = b + a/2 * (a + 5);
write("O valor de b2 e: ");
Write (b2);
}
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste2.txt
Erro na linha: 12
Lexema não esperado [(]
```

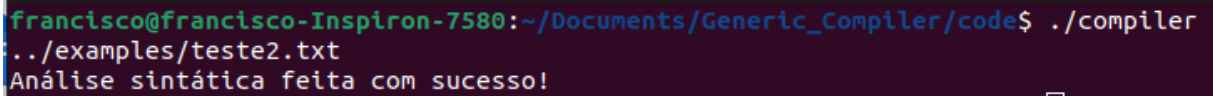
Teste2:

```
class Teste2
/* Teste de comentário
com mais de uma linha */
int a, valor, b_1, b_2;
{
write("Entre com o valor de a: ");
read (a);
b_1 = a * a;
write("O valor de b1 e: ");
write (b_1);
```

```

b_2 = b + a/2 * (a + 5);
write("O valor de b2 e: ");
write (b2);
}

```



```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste2.txt
Análise sintática feita com sucesso!

```

Teste3:

classe Teste3

```

/** Verificando fluxo de controle
Programa com if e while aninhados **/
int i;
int media, soma;
{
soma = 0;
write("Quantos dados deseja informar?" );
read (qtd);
IF (qtd>=2){
i=0;
do{
write("Altura: ");
read (altura);
soma = soma+altura;
i = i + 1;
}while( i < qtd);
media = soma / qtd;
write("Media: ");
write (media);}
else{
write("Quantidade inválida.");
}
}

```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste3.txt
Erro na linha: 01
Lexema não esperado [classe]
```

Teste3:

```
class Teste3
/** Verificando fluxo de controle
Programa com if e while aninhados **/
int i;
int media, soma;
{
soma = 0;
write("Quantos dados deseja informar?" );
read (qtd);
IF (qtd>=2){
i=0;
do{
write("Altura: ");
read (altura);
soma = soma+altura;
i = i + 1;
}while( i < qtd);
media = soma / qtd;
write("Media: ");
write (media);}
else{
write("Quantidade inválida.");
}
}
```

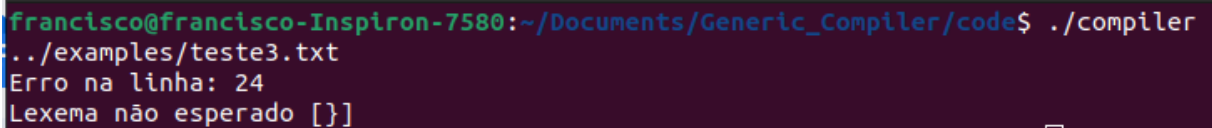
```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste3.txt
Erro na linha: 10
Lexema não esperado [(]
```

Teste3:

```

class Teste3
/** Verificando fluxo de controle
Programa com if e while aninhados **/
int i;
int media, soma;
{
soma = 0;
write("Quantos dados deseja informar?" );
read (qtd);
if (qtd>=2){
i=0;
do{
write("Altura: ");
read (altura);
soma = soma+altura;
i = i + 1;
}while( i < qtd);
media = soma / qtd;
write("Media: ");
write (media);}
else{
write("Quantidade inválida.");
}
}

```



```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
./examples/teste3.txt
Erro na linha: 24
Lexema não esperado [{}]
```

```

class Teste3
/** Verificando fluxo de controle
Programa com if e while aninhados **/
int i;
int media, soma;
{

```

```

soma = 0;
write("Quantos dados deseja informar?" );
read (qtd);
if (qtd>=2){
i=0;
do{
write("Altura: ");
read (altura);
soma = soma+altura;
i = i + 1;
}while( i < qtd);
media = soma / qtd;
write("Media: ");
write (media);}
else{
write("Quantidade inválida.");
};
}

```

```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste3.txt
Análise sintática feita com sucesso!

```

Teste4:

```

{
// Outro programa de teste
int idade, j, k, total;
string nome, texto;
write("Digite o seu nome: ");
read(nome);
write("Digite o seu sobrenome");
read(sobrenome);
write("Digite a sua idade: ");
read (idade);
k = i * (5-i * 50 / 10);
j = i * 10;

```



```

k = i * j / k;
texto = nome + " " + sobrenome + ", os números gerados sao: ";
write (text);
write(j);
write(k);
}

```

```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste4.txt
Erro na linha: 02
Lexema não esperado [{]

```

Teste4:

class Teste4

```

{
// Outro programa de teste
int idade, j, k, total;
string nome, texto;
write("Digite o seu nome: ");
read(nome);
write("Digite o seu sobrenome");
read(sobrenome);
write("Digite a sua idade: ");
read (idade);
k = i * (5-i * 50 / 10);
j = i * 10;
k = i * j / k;
texto = nome + " " + sobrenome + ", os números gerados sao: ";
write (text);
write(j);
write(k);
}

```

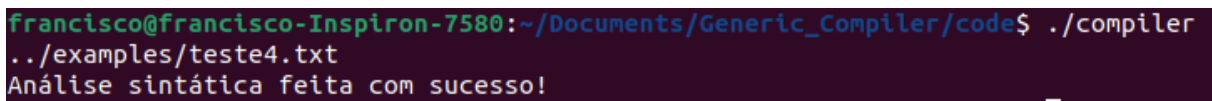
```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste4.txt
Erro na linha: 04
Lexema não esperado [int]

```

Teste4:

```
class Teste4
int idade, j, k, total;
string nome, texto;
{
// Outro programa de teste
write("Digite o seu nome: ");
read(nome);
write("Digite o seu sobrenome");
read(sobrenome);
write("Digite a sua idade: ");
read (idade);
k = i * (5-i * 50 / 10);
j = i * 10;
k = i * j / k;
texto = nome + " " + sobrenome + ", os números gerados sao: ";
write (text);
write(j);
write(k);
}
```

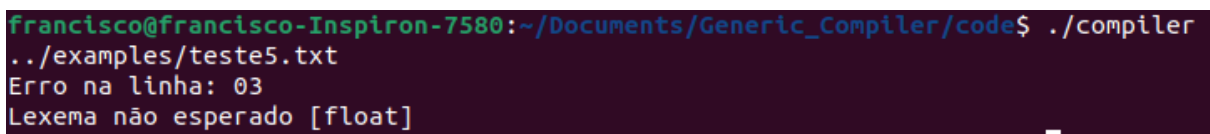
A terminal window with a dark background and light green text. The prompt is 'Francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code\$'. The user has entered './compiler ../examples/teste4.txt'. The output is 'Análise sintática feita com sucesso!' followed by a cursor.

```
Francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste4.txt
Análise sintática feita com sucesso!
```

Teste5:

```
class MinhaClasse
{
float a, b, c;
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
```

```
if ( a>b && a>c )
maior = a;
else
if (b>c)
maior = b;
else
maior = c;
write("O maior número é: ");
write(maior);
```

A terminal window with a dark background. The prompt is 'francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code\$'. The user has entered './compiler ../examples/teste5.txt'. The output shows an error: 'Erro na linha: 03' and 'Lexema não esperado [float]'.

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 03
Lexema não esperado [float]
```

Teste5:

```
class MinhaClasse
float a, b, c;
{
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( a>b && a>c )
maior = a;
else
if (b>c)
maior = b;
else
maior = c;
write("O maior número é: ");
write(maior);
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 09
Lexema não esperado [;]
```

Teste5:

class MinhaClasse

float a, b, c;

{

write("Digite um número");

read(a);

write("Digite outro número: ");

read(b);

write("Digite mais um número: ");

read(c);

maior = 0;

if (a>b && a>c)

maior = a;

else

if (b>c)

maior = b;

else

maior = c;

write("O maior número é: ");

write(maior);

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 11
Lexema não esperado [>]
```

Teste5:

class MinhaClasse

float a, b, c;

{

write("Digite um número");

read(a);

write("Digite outro número: ");

```

read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( a>b && a>c )
maior = a;
else
if (b>c)
maior = b;
else
maior = c;
write("O maior número é: ");
write(maior);

```

```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 12
Lexema não esperado [maior]

```

Teste5:

```

class MinhaClasse
float a, b, c;
{
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( (a>b) && (a>c) )
maior = a;
else
if (b>c)
maior = b;
else
maior = c;

```

```
write("O maior número é: ");  
write(maior);
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler  
../examples/teste5.txt  
Erro na linha: 14  
Lexema não esperado [if]
```

Teste5:

```
class MinhaClasse
```

```
float a, b, c;
```

```
{
```

```
write("Digite um número");
```

```
read(a);
```

```
write("Digite outro número: ");
```

```
read(b);
```

```
write("Digite mais um número: ");
```

```
read(c);
```

```
maior = 0;
```

```
if ( (a>b) && (a>c) ) {
```

```
maior = a;
```

```
} else
```

```
if (b>c)
```

```
maior = b;
```

```
else
```

```
maior = c;
```

```
write("O maior número é: ");
```

```
write(maior);
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler  
../examples/teste5.txt  
Erro na linha: 14  
Lexema não esperado [if]
```

Teste5:

```
class MinhaClasse
```

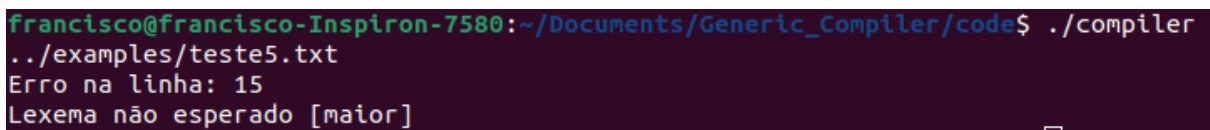
```
float a, b, c;
```

```
{
```

```

write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( (a>b) && (a>c) ) {
maior = a;
} else {
if (b>c)
maior = b;
else
maior = c;
write("O maior número é: ");
write(maior);
}

```



```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 15
Lexema não esperado [maior]

```

Teste5:

```

class MinhaClasse
float a, b, c;
{
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( (a>b) && (a>c) ) {
maior = a;
} else {

```

```

if (b>c) {
maior = b;
else
maior = c;
write("O maior número é: ");
write(maior);
}
}

```

```

francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 16
Lexema não esperado [else]

```

Teste5:

```

class MinhaClasse
float a, b, c;
{
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( (a>b) && (a>c) ) {
maior = a;
} else {
if (b>c) {
maior = b;
else {
maior = c;
write("O maior número é: ");
write(maior);
}
}
}
}

```



```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Erro na linha: 21
Lexema não esperado [}]
```

```
class MinhaClasse
float a, b, c;
{
write("Digite um número");
read(a);
write("Digite outro número: ");
read(b);
write("Digite mais um número: ");
read(c);
maior = 0;
if ( (a>b) && (a>c) ) {
maior = a;
} else {
if (b>c) {
maior = b;
else {
maior = c;
write("O maior número é: ");
write(maior);
};
};
}
```

```
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$ ./compiler
../examples/teste5.txt
Análise sintática feita com sucesso!
francisco@francisco-Inspiron-7580:~/Documents/Generic_Compiler/code$
```