

Analizador Sintático - Relatório

Nome: Pedro Santos Oliveira

Matrícula: 20203001340

Disciplina: Compiladores

Linguagem de Programação: C++

1 - Forma de uso do Compilador

- O primeiro passo é definir o caminho do arquivo de teste dentro do construtor da classe Lexer, presente no arquivo “Lexer.cpp” na linha 10

```
// Inicializar leitor de arquivo
file.open("Testes\\Teste1.txt");
```

- O segundo passo é ajustar os comandos de include nos arquivos **.cpp**, tanto o arquivo Main.cpp quanto os arquivos contidos nas pastas Lexer e Parser. Os comandos de include que estão incluindo outros arquivos atualmente estão com os caminhos relativos dos arquivos na minha máquina para possibilitar a compilação local. Para conseguir compilar, será necessário alterar esse caminho para o caminho relativo dos arquivos na máquina em que o programa será compilado. Segue um exemplo de como estão os comandos de include no arquivo Main.cpp para possibilitar a compilação na minha máquina:

```
#include <bits/stdc++.h>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Lexer\Tag.h>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Lexer\Token.cpp>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Lexer\Num.cpp>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Lexer\Real.cpp>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Lexer\Word.cpp>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Lexer\Lexer.cpp>
#include <C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho\Parser\Parser.cpp>
```

- Em seguida, resta apenas compilar o arquivo Main.cpp
- Para compilar o arquivo Main.cpp basta ir para o terminal e digitar “g++ Main.cpp”
- Após isso, um arquivo executável será criado com o nome “a.exe”
- Para rodar o arquivo a.exe basta ir para o terminal e digitar “./a.exe”
- Assim, o programa do compilador será executado utilizando o teste especificado e as supostas saídas do programa serão apresentadas no terminal
- Para cada arquivo de teste é necessário alterar o caminho do arquivo no construtor da classe Lexer em “Lexer.cpp” e compilar o programa novamente seguindo os passos acima.

2 – Descrição da abordagem utilizada no Compilador

Abordagem Analisador Sintático

- A abordagem utilizada na construção do Analisador Sintático foi a abordagem LL(1), ou seja, uma abordagem de um Analisador Sintático Descendente. Esta abordagem só pode ser aplicada em gramáticas que não possuem recursão à esquerda e que não possuem prefixos comuns, sendo assim, para aplicar esta abordagem realizamos pequenas alterações na gramática para que ela se tornasse uma gramática LL(1). Por fim, a partir da abordagem utilizada foi construído um método para cada não terminal da gramática e outros dois métodos, responsáveis pela conferência do Token recebido, e pela leitura do próximo Token da sequência. Para implementar essa abordagem foram seguidos os passos dados nos Slides de Análise Sintática disponibilizados no MOODLE.

Principais Classes do Compilador

- **Class Parser (Parser)**
 - A classe Parser é a classe que representa o Analisador Sintático, portanto, a classe principal dessa etapa do trabalho. Na classe Parser teremos um atributo que referencia o Analisador Léxico instanciado no método main, e um atributo do tipo Token para receber os Tokens formados pelo Analisador Léxico. Além disso teremos também um método para checar o Token recebido com o Token esperado pela gramática, um método para “comer” o Token lido e avançar na leitura, um método para organizar os prints de erros do Analisador Sintático, um método “init” que terá o papel de inciar a análise dos Tokens, e os métodos construtores da classe. Por fim, além dos métodos já citados, o Analisador Sintático também possuirá um método para cada símbolo não terminal presente na gramática da linguagem como explicado acima na descrição da abordagem utilizada.
- **Class Main**
 - A classe Main é a classe que usaremos para instanciar o Analisador Léxico e o Analisador Sintático e chamar o método init() para inicializar a análise.

Principais Métodos do Compilador

- **Parser.cpp → advance()**
 - O método advance() é responsável por chamar o método scan() do Analisador Léxico, ou seja, responsável por coletar o próximo Token da sequência e salvá-lo na variável “tok” do Parser. Além disso, na hora que o Token é coletado esse método faz algumas pequenas checagens para analisar o tipo do Token coletado.
- **Parser.cpp → eat(int t)**
 - O método eat(int t) é responsável por fazer a conferência do Token coletado da sequência com o Token que era esperado ser coletado da sequência, baseando-se nas regras da gramática da linguagem. Assim, o método realiza a checagem e caso ela não seja positiva é retornado um erro sintático.
- **Parser.cpp → error(vector<int> expected)**
 - O método error() será responsável por organizar e realizar os prints de erros sintáticos, sempre printando o Token recebido e o(s) Token(s) esperado(s).
- **Parser.cpp → init()**
 - O método init() será responsável por inicializar a execução do Analisador Sintático chamando os métodos advance() e program(), e também responsável por checar se o último Token lido possui Tag referente a Tag de um Token EOF.
- **Parser.cpp → void program() | void decllist() | void decl() | void identlist() | void type() | void stmtlist() | void stmt() | void assignstmt() | void condition() | void repeatstmt() | void stmtsuffix() | void whilestmt() | void stmtprefix() | void readstmt() | void writestmt() | void writable() | void factora() | void factor() | void relop() | void addop() | void mulop() | void constant() | void ifstmtprime() | void ifstmt() | void expressionprime() | void expression() | void simpleexprprime() | void simpleexpr() | void termprime() | void term() (Parser)**
 - Os métodos listados acima são referentes a cada um dos símbolos não terminais da gramática, e cada um deles implementa a sequência de Tokens para cada produção de cada um dos símbolos não terminais.

- **Tabelas de Tags**
 - Mantive a Tabela de Tags nesse relatório para facilitar a análise dos prints da execução, uma vez que os prints de erros sintáticos irão conter as tags dos respectivos Tokens esperados e dos respectivos Tokens recebidos.

- **Tags dos operadores:**

LEXEMA	TIPO DA TAG	VALOR DA TAG
“==”	EQ	256
“>”	GT	257
“> =”	GE	258
“<”	LT	259
“< =”	LE	260
“! =”	NE	261
“ ”	OR	262
“&&”	AND	263

- **Tags das palavras reservadas:**

LEXEMA	TIPO DA TAG	VALOR DA TAG
“program”	PROGRAM	264
“begin”	BEGIN	265
“end”	END	266
“is”	IS	267
“int”	INT	268
“float”	FLOAT	269
“char”	CHAR	270
“if”	IF	271
“then”	THEN	272
“else”	ELSE	273
“repeat”	REPEAT	274
“until”	UNTIL	275
“while”	WHILE	276
“do”	DO	277
“read”	READ	278
“write”	WRITE	279

- **Demais Tags:**

LEXEMA	TIPO DA TAG	VALOR DA TAG
Valor Numérico Inteiro	NUM	280
Valor Numérico Float	REAL	281
Identificador	ID	282
	_EOF	283
Valor de char_const	CHAR_CONST	284
Valor de literal	LITERAL	285
	LEXICAL_ERROR	500
	ERROR_TO_OPEN_FILE	501

3 – Mudanças na Gramática

- Regras com Prefixo Comum

if-stmt ::= if condition then stmt-list end
| if condition then stmt-list else stmt-list end

expression ::= simple-expr | simple-expr relop simple-expr

- Regras com Recursão à Esquerda

simple-expr ::= term | simple-expr addop term

term ::= factor-a | term mulop factor-a

- Regras Modificadas

if-stmt-prime	→ if condition then stmt-list if-stmt
if-stmt	→ end else stmt-list end
expression-prime	→ simple-expression expression
expression	→ relop simple-expr λ
simple-expr	→ term simple-expr-prime
simple-expr-prime	→ addop term simple-expr-prime λ
term	→ factor-a term-prime
term-prime	→ mulop factor-a term-prime λ

4 – Resultado dos testes realizados

Testes Parser

Os testes utilizados nessa etapa do trabalho serão os testes utilizados na primeira etapa porém sem os erros léxicos identificados na etapa anterior. Para fins de facilitar a análise do funcionamento do programa, sempre que ocorrer um erro sintático será printado no terminal uma frase indicando o erro, a linha onde o erro ocorreu, a(s) Tag(s) do(s) Token(s) esperado(s) pelo Parser, a Tag do Token lido pelo Parser, e por fim, a função que realizou o print do erro, com fins de facilitar a análise do código. Apenas as funções error() e eat() são responsáveis por printar possíveis erros sintáticos.

- TesteParser1.txt (Parser)

Programa Fonte (1):

```
programa teste1

a, b is int;
result is int;
a,x is float;
begin
a = 12a;
x = 12.1;
read (a);
```

```
read (b);
read (c)
result = (a*b + 1) / (c+2);
write {Resultado: };
write (result);
end.
```

Saída Esperada (1): A saída esperada para o teste acima é um erro sintático na primeira linha, pois a primeira palavra do teste é um identificador (**ID**) de Tag 282, enquanto deveria ser a palavra reservada **program** de Tag 264, definido na primeira regra da gramática.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 1 -> Expected tag(s) 264 but found tag 282 : (error())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (2):

```
program teste1

a, b is int;
result is int;
a,x is float;
begin
a = 12a;
x = 12.1;
read (a);
read (b);
read (c)
result = (a*b + 1) / (c+2);
write {Resultado: };
write (result);
end.
```

Saída Esperada (2): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na sétima linha, no comando de atribuição “**a = 12a**”, pois o comando de atribuição em questão (assign-stmt) é finalizado quando o valor 12 (integer const) é lido, e como o Token na sequência é um identificador de Tag 282 e de lexema “**a**” um erro ocorre uma vez que o Token esperado seria um Token referente a palavra reservada **end** de Tag 266 definido na primeira regra da gramática.

Saída do Compilador (2):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 7 -> Expected tag(s) 266 but found tag 282 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (3):

```
program teste1

a, b is int;
result is int;
a,x is float;
begin
a = 12;
x = 12.1;
read (a);
read (b);
read (c)
result = (a*b + 1) / (c+2);
write {Resultado: };
write (result);
end.
```

Saída Esperada (3): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na linha onze, no comando “**read (c)**”, pois como não tem um “;” após o comando de **read** o Token esperado seria um Token referente a palavra reservada **end** de Tag 266 definido na primeira regra da gramática. Portanto, o print de erro aponta um erro na linha doze, que é a linha subsequente, uma vez que o Parser entende que o identificador “**result**” está ocupando o lugar do que deveria ser um “;”.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 12 -> Expected tag(s) 266 but found tag 282 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (4):

```
program teste1

a, b is int;
result is int;
a,x is float;
begin
a = 12;
x = 12.1;
read (a);
read (b);
read (c);
result = (a*b + 1) / (c+2);
write {Resultado: };
write (result);
end.
```

Saída Esperada (4): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na linha treze, no comando “**write {Resultado: }**”, pois após o comando **write** é esperado um “(” de Tag 40, porém é encontrado um Token do tipo literal de Tag 285.

Saída do Compilador (4):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 13 -> Expected tag(s) 40 but found tag 285 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (5):

```
program teste1

a, b is int;
result is int;
a,x is float;
begin
a = 12;
x = 12.1;
read (a);
read (b);
read (c);
result = (a*b + 1) / (c+2);
write ( {Resultado: } );
write (result);
end.
```

Saída Esperada (5): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na última linha, uma vez que o último comando de **write** termina com um “;” indicando que um outro **stmt** viria na seqência, porém quando é lido a palavre reservada **end** de Tag 266 ocorre um erro sintático.

Saída do Compilador (5):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 15 -> Expected tag(s) 282 271 276 274 278 279 but found tag 266 : (error())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (6):

```
program teste1

a, b is int;
result is int;
a,x is float;
begin
a = 12;
x = 12.1;
read (a);
read (b);
read (c);
result = (a*b + 1) / (c+2);
write ( {Resultado: } );
write (result)
end.
```

Saída Esperada (6): Após a correção do erro acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (6):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Analysis Complete !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- **TesteParser2.txt (Parser)**

Programa Fonte (1):

```
program teste2
a, b, c:int;
d, _var: float;
teste2 = 1;
Read (a);
b = a * a;
c = b + a/2 * (35/b);
write c;
val := 34.2
c = val + 2.2 + a;
write (val)
end.
```

Saída Esperada (1): A saída esperada para o teste acima é um erro sintático na linha dois, uma vez que após o identificador “**c**” é encontrado um Token com lexema igual a “:” de Tag 58, enquanto o esperado pela gramática seria um Token referente a palavra reservada **is** de Tag 267. O mesmo erro sintático ocorrerá na linha três.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 2 -> Expected tag(s) 267 but found tag 58 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (2):

```
program teste2
a, b, c is int;
d, _var is float;
teste2 = 1;
Read (a);
b = a * a;
c = b + a/2 * (35/b);
write c;
val := 34.2
c = val + 2.2 + a;
write (val)
end.
```

Saída Esperada (2): Após a correção do erro acima, em todas as linhas citadas na análise anterior, a saída esperada para o teste acima é um erro sintático na linha três, uma vez que o Parser estava esperando um identificador de Tag 282 e recebeu um Token com lexema igual a “_” de Tag 95. Como o identificador não pode começar com “_” então ocorre um erro. O erro em questão pode ser considerado um erro léxico, porém o Analisador Léxico implementado reconhece o “_” como um Token de lexema “_” e o “**var**” como um Token de lexema “**var**”.

Saída do Compilador (2):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 3 -> Expected tag(s) 282 but found tag 95 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (3):

```
program teste2
a, b, c is int;
d, var is float;
teste2 = 1;
Read (a);
b = a * a;
c = b + a/2 * (35/b);
write c;
val := 34.2
c = val + 2.2 + a;
write (val)
end.
```

Saída Esperada (3): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na linha quatro, uma vez que o Parser vai entender que a linha quatro está dentro de uma produção de **decl-list** enquanto na verdade é uma produção de **stmt-list**, isso acontece pois não existe a palavra **begin** antes do comando de atribuição e depois da declaração das variáveis.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 4 -> Expected tag(s) 267 but found tag 61 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```


Programa Fonte (4):

```
program teste2
  a, b, c is int;
  d, var is float;
begin
  teste2 = 1;
  Read (a);
  b = a * a;
  c = b + a/2 * (35/b);
  write c;
  val := 34.2
  c = val + 2.2 + a;
  write (val)
end.
```

Saída Esperada (4): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na linha seis, por conta de a palavra “**Read**” estar escrita com “R” maiúsculo o Lexer entende que o Token referente a esse lexema será um Token do tipo identificador, e por conta disso o Parser entende que está lendo uma produção de **assign-stmt** e espera que o próximo Token analisado seja um Token com lexema igual a “=” e Tag 61, enquanto na verdade deveria estar lendo um **read-stmt**.

Saída do Compilador (4):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 6 -> Expected tag(s) 61 but found tag 40 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (5):

```
program teste2
  a, b, c is int;
  d, var is float;
begin
  teste2 = 1;
  read (a);
  b = a * a;
  c = b + a/2 * (35/b);
  write c;
  val := 34.2
  c = val + 2.2 + a;
  write (val)
end.
```

Saída Esperada (5): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na linha nove, uma vez que o Parser está analisando um produção de **write-stmt** e espera que após a palavra reservada **write** venha um Token com lexema igual a “(” e Tag 40, enquanto na verdade vem um Token do tipo identificador de Tag 282.

Saída do Compilador (5):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 9 -> Expected tag(s) 40 but found tag 282 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (6):

```
program teste2
  a, b, c is int;
  d, var is float;
begin
  teste2 = 1;
  read (a);
```

```
b = a * a;  
c = b + a/2 * (35/b);  
write (c);  
val := 34.2  
c = val + 2.2 + a;  
write (val)  
end.
```

Saída Esperada (6): Após a correção do erro acima, a saída esperada para o teste acima é um erro sintático na linha dez, uma vez que o Parser está analisando uma produção de **assign-stmt** e estava esperando um Token de lexema igual a “=” e Tag 58, enquanto recebeu um Token de lexema igual a “:” e Tag 61. Além desse erro, a linha dez não possui um “;” no final da linha, uma vez que na linha seguinte contém um comando de atribuição, a falta desse “;” também causaria um erro sintático pois o Parser estaria esperando a palavra reservada **end**.

Saída do Compilador (6):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe  
Parser Started...  
Syntactical Error in line 10 -> Expected tag(s) 61 but found tag 58 : (eat())  
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (7):

```
program teste2  
a, b, c is int;  
d, var is float;  
begin  
teste2 = 1;  
read (a);  
b = a * a;  
c = b + a/2 * (35/b);  
write (c);  
val = 34.2;  
c = val + 2.2 + a;  
write (val)  
end.
```

Saída Esperada (7): Após a correção dos erros acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (7):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe  
Parser Started...  
Syntactical Analysis Complete !!!  
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- TesteParser3.txt (Parser)

Programa Fonte (1):

```
program  
a, aux is int;  
b is float  
begin  
b = 0;  
int (a);  
int(b);  
if (a>b) then /*troca variaveis*/  
aux = b;  
b = a;
```

```
a = aux
end;
write(a;
write(b)
```

Saída Esperada (1): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha dois, uma vez que após a palavra reservada **program** é esperado um identificador seguido de uma produção de **decl-list**, e a produção **decl-list** começa com um identificador, porém o Lexer não reconhece “,” como identificador, portanto o Parser entende que não deve processar uma produção de **decl-list** e espera pela palavra reservada **begin**, seguindo a lógica da primeira regra da gramática.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 2 -> Expected tag(s) 265 but found tag 44 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (2):

```
program teste2
a, aux is int;
b is float
begin
b = 0;
int (a);
int(b);
if (a>b) then /*troca variaveis*/
aux = b;
b = a;
a = aux
end;
write(a;
write(b)
```

Saída Esperada (2): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha quatro, uma vez que o Parser está analisando uma produção de decl-list que não termina com “,” de Tag 59, portanto ele recebe um Token referente a palavra reservada **begin** de Tag 265 e retorna um erro sintático.

Saída do Compilador (2):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 4 -> Expected tag(s) 59 but found tag 265 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (3):

```
program teste2
a, aux is int;
b is float;
begin
b = 0;
int (a);
int(b);
if (a>b) then /*troca variaveis*/
aux = b;
b = a;
a = aux
end;
write(a;
write(b)
```

Saída Esperada (3): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha seis, uma vez que o Parser está analisando uma produção de **stmt-list**, e nenhuma produção de stmt-list começa com o Token referente a palavra reservada **int**, portanto é retornado um erro. Além disso, na linha sete ocorre o mesmo erro.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 6 -> Expected tag(s) 282 271 276 274 278 279 but found tag 268 : (error())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (4):

```
program teste2
a, aux is int;
b is float;
begin
b = 0;
if (a>b) then /*troca variaveis*/
aux = b;
b = a;
a = aux
end;
write(a;
write(b)
```

Saída Esperada (4): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha onze, uma vez que o Parser está lendo uma produção de **write-stmt** mal formada, pois não ocorre o fechamento do parêntesis. O Lexer não identifica isso como um erro léxico pois identifica o parêntesis como um Token separado do resto da produção.

Saída do Compilador (4):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 11 -> Expected tag(s) 41 but found tag 59 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (5):

```
program teste2
a, aux is int;
b is float;
begin
b = 0;
if (a>b) then /*troca variaveis*/
aux = b;
b = a;
a = aux
end;
write(a);
write(b)
```

Saída Esperada (5): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha doze, uma vez que o Parser termina de analisar uma produção de **stmt-list** e espera receber um Token referente a palavra reservada **end** de Tag 266 mas encontra um Token de Tag 283 referente a EOF.

Saída do Compilador (5):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 12 -> Expected tag(s) 266 but found tag 283 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (6):

```
program teste2
  a, aux is int;
  b is float;
  begin
    b = 0;
    if (a>b) then /*troca variaveis*/
      aux = b;
      b = a;
      a = aux
    end;
    write(a);
    write(b)
  end.
```

Saída Esperada (6): Após a correção dos erros acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (6):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Analysis Complete !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- TesteParser4.txt (Parser)

Programa Fonte (1):

```
programa teste4
/* Teste4 do meu compilador
pontuacao, pontuacaoMaxima, disponibilidade is inteiro;
pontuacaoMinima is char;
begin
  pontuacaoMinima = 50;
  pontuacaoMaxima = 100;
  write({Pontuacao do candidato: });
  read(pontuacao);
  write({Disponibilidade do candidato: });
  read(disponibilidade);

  while (pontuacao>0 & (pontuacao<=pontuacaoMaxima) do
    if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
      write({Candidato aprovado.})
    else
      write({Candidato reprovado.})
    end
    write({Pontuacao do candidato: });
    read(pontuacao);
    write({Disponibilidade do candidato: });
    read(disponibilidade);
  end
end*/
```

Saída Esperada (1): A saída esperada para o teste acima é um erro sintático na linha um, uma vez que o Parser espera um Token referente a palavra reservada **program** de Tag 264 e recebe um Token referente a um identificador de Tag 282.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 1 -> Expected tag(s) 264 but found tag 282 : (error())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (2):

```
program teste4
/* Teste4 do meu compilador
pontuacao, pontuacaoMaxima, disponibilidade is inteiro;
pontuacaoMinima is char;
begin
pontuacaoMinima = 50;
pontuacaoMaxima = 100;
write({Pontuacao do candidato: });
read(pontuacao);
write({Disponibilidade do candidato: });
read(disponibilidade);

while (pontuacao>0 & (pontuacao<=pontuacaoMaxima) do
if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
write({Candidato aprovado.})
else
write({Candidato reprovado.})
end
write({Pontuacao do candidato: });
read(pontuacao);
write({Disponibilidade do candidato: });
read(disponibilidade);
end
end*/
```

Saída Esperada (2): A saída esperada para o teste acima é um erro sintático apontado na linha três, pois o Parser espera o resto da produção da primeira regra da gramática que seria: **begin** stmt-list **end** “.”, porém ele não encontra, ao invés disso ele encontra um Token referente a EOF de Tag 283. Para que esse teste passe a funcionar é necessário acrescentar o restante do corpo dessa produção.

Saída do Compilador (2):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 3 -> Expected tag(s) 265 but found tag 283 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (3):

```
program teste4
begin
read (a)
end.
/* Teste4 do meu compilador
pontuacao, pontuacaoMaxima, disponibilidade is inteiro;
pontuacaoMinima is char;
begin
pontuacaoMinima = 50;
pontuacaoMaxima = 100;
write({Pontuacao do candidato: });
read(pontuacao);
write({Disponibilidade do candidato: });
read(disponibilidade);

while (pontuacao>0 & (pontuacao<=pontuacaoMaxima) do
if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
write({Candidato aprovado.})
else
write({Candidato reprovado.})
end
write({Pontuacao do candidato: });
read(pontuacao);
write({Disponibilidade do candidato: });
read(disponibilidade);
```

```
end
end*/
```

Saída Esperada (3): Após a correção dos erros acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Analysis Complete !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

- TesteParser5.txt (Parser)

Programa Fonte (1):

```
/* Teste do meu compilador */
program teste5
  a, b, c, maior is int;
  outro is char;
begin
  repeat
    write({A});
    read(a);
    write({B});
    read(b);
    write({C});
    read(c);
    if ( (a>b) && (a>c) ) end
    maior = a

  else
    if (b>c) then
      maior = b;

  else
    maior = c
  end
end;
write({Maior valor:});
write (maior);
write ({Outro? (S/N)});
read(outro);
until (outro == 'N' || outro == 'n')
end.
```

Saída Esperada (1): A saída esperada para o teste acima é um erro sintático na linha treze, uma vez que o Parser está analisando uma produção de **if-stmt** e recebe um Token referente a palavra reservada **end** de Tag 266 enquanto esperava por um Token referente a palavra reservada **then** de Tag 272.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 13 -> Expected tag(s) 272 but found tag 266 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

Programa Fonte (2):

```
/* Teste do meu compilador */
program teste5
```

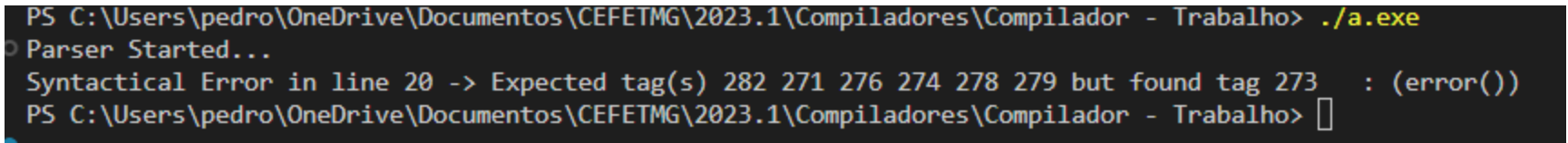
```
a, b, c, maior is int;
outro is char;
begin
  repeat
    write({A});
    read(a);
    write({B});
    read(b);
    write({C});
    read(c);
    if ( (a>b) && (a>c) ) then
      maior = a

    else
      if (b>c) then
        maior = b;

      else
        maior = c
      end
    end;
    write({Maior valor:});
    write (maior);
    write ({Outro? (S/N)});
    read(outro);
    until (outro == 'N' || outro == 'n')
  end.
```

Saída Esperada (2): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha vinte, uma vez que após o comando de atribuição tem um “;” fazendo com que o Parser espere por mais uma produção de **stmt**, enquanto na verdade ele recebe um Token referente a palavra reservada **else** que não inicializa nenhuma produção de **stmt**. O mesmo erro acontece na linha vinte e oito, pois o comando **read** na linha vinte e sete termina com um “;”, e portanto o comando **until** da linha seguinte não é esperado pelo Parser.

Saída do Compilador (2):



Programa Fonte (3):

```
/* Teste do meu compilador */
program teste5
  a, b, c, maior is int;
  outro is char;
begin
  repeat
    write({A});
    read(a);
    write({B});
    read(b);
    write({C});
    read(c);
    if ( (a>b) && (a>c) ) then
      maior = a

    else
      if (b>c) then
        maior = b

      else
        maior = c
      end
    end;
    write({Maior valor:});
    write (maior);
    write ({Outro? (S/N)});
    read(outro)
    until (outro == 'N' || outro == 'n')
  end.
```


Saída Esperada (3): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha vinte e oito, uma vez que o Parser está analisando uma produção de **factor**, ele espera ler um Token de lexema “)” e Tag 41 após o identificador, porém ele recebe um Token de Tag 256 referente ao símbolo “==”. Para concertar esse probelma é necessário colocar entre parêntesis as condições do comando **until**.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 28 -> Expected tag(s) 41 but found tag 256 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (4):

```
/* Teste do meu compilador */
program teste5
  a, b, c, maior is int;
  outro is char;
begin
  repeat
    write({A});
    read(a);
    write({B});
    read(b);
    write({C});
    read(c);
    if ( (a>b) && (a>c) ) then
      maior = a

    else
      if (b>c) then
        maior = b

      else
        maior = c
      end
    end;
    write({Maior valor:});
    write (maior);
    write ({Outro? (S/N)});
    read(outro)
    until ( (outro == 'N') || (outro == 'n') )
  end.
```

Saída Esperada (4): Após a correção dos erros acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (4):

```
● PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
● Parser Started...
  Syntactical Analysis Complete !!!
○ PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- **TesteParser6.txt (Parser)**

Programa Fonte (1):

```
/* Teste */
program teste6

  x, y is int;

  duda is char;
```

```
/* comentário 1*/

begin
  repeat
    write(x);
    read(x);

    if (x) then
      maior = a;
    end

/* comentário 2*/

end
```

Saída Esperada (1): A saída esperada para o teste acima é um erro sintático na linha dezessete, uma vez que a linha anterior é referente a uma produção de **stmt** e termina com “;”, ou seja, o Parser espera receber mais uma produção de **stmt** na sequência, mas na verdade recebe um Token referente a palavra reservada **end**.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 17 -> Expected tag(s) 282 271 276 274 278 279 but found tag 266 : (error())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (2):

```
/* Teste */
program teste6

  x, y is int;

  duda is char;

/* comentário 1*/

begin
  repeat
    write(x);
    read(x);

    if (x) then
      maior = a
    end

/* comentário 2*/

end
```

Saída Esperada (2): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha vinte e um, uma vez que o Parser está analisando uma produção de **repeat-stmt** e está esperando por um **stmt-suffix**, enquanto na verdade ele recebe um Token referente a palavra reservada **end** que não faz parte de nenhuma produção de **stmt-suffix**.

Saída do Compilador (2):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 21 -> Expected tag(s) 275 but found tag 266 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (3):

```
/* Teste */
program teste6

  x, y is int;

  duda is char;

/* comentário 1*/

begin
  repeat
    write(x);
    read(x);

    if (x) then
      maior = a
    end

/* comentário 2*/

until(x > 10)
end.
```

Saída Esperada (3): Após a correção dos erros acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Analysis Complete !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- TesteParser7.txt (Parser)

Programa Fonte (1):

```
program teste7
program teste

  a, b is int; /*variavel a*/ /*varialvel b*/
  c is float; /*variavel c*/
begin
  a = 1;
  c = 1.0;
  read( a );

  c = b + 5.2;

  write {c: };
  write (c);

end
```

Saída Esperada (1): A saída esperada para o teste acima é um erro sintático na linha dois, uma vez que o Parser esperava por receber um Token referente a palavra reservad **begin** de Tag 265 mas na verdade recebe um Token referente a palavra reservada **program** de Tag 264.

Saída do Compilador (1):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> g++ Main.cpp
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 2 -> Expected tag(s) 265 but found tag 264 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (2):

```
program teste7

  a, b is int; /*variavel a*/ /*varialvel b*/
  c is float; /*variavel c*/
begin
  a = 1;
  c = 1.0;
  read( a );

  c = b + 5.2;

  write {c: };
  write (c);

end
```

Saída Esperada (2): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha doze, uma vez que o Parser está analisando uma produção de **write-stmt** e espera receber um Token de lexema “(” e Tag 40, enquanto na verdade recebe um Token referente a um literal de Tag 285.

Saída do Compilador (2):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 12 -> Expected tag(s) 40 but found tag 285 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (3):

```
program teste7

  a, b is int; /*variavel a*/ /*varialvel b*/
  c is float; /*variavel c*/
begin
  a = 1;
  c = 1.0;
  read( a );

  c = b + 5.2;

  write ({c: });
  write (c);

end
```

Saída Esperada (3): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha quize, uma vez que o Parser espera analisar mais uma produção de **stmt** já que a produção anterior termina com “;”, porém ele receb um Token referente a palavra reservada **end** de Tag 266.

Saída do Compilador (3):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 15 -> Expected tag(s) 282 271 276 274 278 279 but found tag 266 : (error())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (4):

```
program teste7
```

```
a, b is int; /*variavel a*/ /*varialvel b*/
c is float; /*variavel c*/
begin
a = 1;
c = 1.0;
read( a );

c = b + 5.2;

write ({c: });
write (c)

end
```

Saída Esperada (4): Após a correção dos erros acima, a saída esperada para o teste acima é um erro sintático na linha quize, uma vez que o Parser espera receber um Token de lexema “.” e Tag 46, enquanto na verdade ele recebe um Token que sinaliza EOF de Tag 283.

Saída do Compilador (4):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Error in line 15 -> Expected tag(s) 46 but found tag 283 : (eat())
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (5):

```
program teste7

a, b is int; /*variavel a*/ /*varialvel b*/
c is float; /*variavel c*/
begin
a = 1;
c = 1.0;
read( a );

c = b + 5.2;

write ({c: });
write (c)

end.
```

Saída Esperada (5): Após a correção dos erros acima, a saída esperada para o teste acima é uma saída sem erros, indicando uma Análise Sintática completa.

Saída do Compilador (5):

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
Parser Started...
Syntactical Analysis Complete !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```