Compilador
Analisador Léxico

Heloisa Toledo
Leonardo Tereza
Marcus Buzette

Belo Horizonte
2018.

Conteúdo

1 Entrega do Analisador Léxico

1.1 Introdução

Compilador é um processador de linguagens. Um programa que traduz um programa escrito em uma linguagem (lingiagem fonte) e gera um programa equivalente escrito em outra linguagem (linguagem alvo ou linguagem obejto).

Classicamente, um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina, específica para um processador e sistema operacional.

1.1.1 Tabela de Símbolos

É uma estrutura de dados gerada pelo compilador com o objetivo de armazenar informações sobre os nomes (identificadores de variáveis, de parâmetros, de funções, de procedimentos, etc) definidos no programa fonte. Essa tabela associa atributos (tipo, escopo, limites no caso de vetores e número de parâmetros no caso de funções) aos nomes que foram definidos pelo programador. A construção dessa tabela, em geral, se dá durante a análise léxica, quando os identificadores são reconhecidos.

1.1.2 Analisador Léxico

É a primeira fase do compilador. A função do analisador léxico, também denominado scanner, é ler o código-fonte, caractere a caractere, buscando a separação e identificação dos elementos componentes do programa-fonte, denominados símbolos léxicos ou tokens.

Os tokens constituem classes de símbolos tais como palavras reservadas, delimitadores, identificadores, etc., e podem ser representados, internamente, através do próprio símbolo (como no caso dos delimitadores e das palavras reservadas) ou por um par ordenado, no qual o primeiro elemento indica a classe do símbolo, e o segundo, um índice para uma área onde o próprio símbolo foi armazenado (por exemplo, um identificador e sua entrada numa tabela de identifidadores).

1.1.3 Analisador Sintático

É a segunda fase do compilador e também é conhecida como Parser. O analisador sintático, de forma geral, é responsável por verificar a ordem dos tokens reconhecidos pelo Scanner.

O parser tem como entrada a saida do analisador léxico por meio do método getToken(). O analisador sintático organiza todos os tokens em uma estrutura de dados, e, nesse caso, foi utilizado uma ávore de derivação seguindo as regras da gramatica.

1.2 Forma de Uso do Compilador

O projeto foi desenvolvido no Netbeans. Para isso, deve-se baixar o arquivo compactado, descompactar, abrir o projeto no Netbeans e rodar.

Todos os testes estão na pasta raíz do projeto. Bastando que altere o nome que se encontra no arquivo TPCompiladores.java.

Os arquivos foram nomiados de acordo com a especificação do trabalho. Sendo eles: "teste1.txt", "teste2.txt", "teste3.txt", "teste4.txt", "teste5.txt"e "teste6.txt".

1.3 Descrição da Abordagem Utilizada

Principais Classes

1.3.1 Lexer.java

Classe que implementa o analisador léxico. Seu constrututor insere as palavras reservadas na tabela de símbolos. Possui um método scan que devolve um Token.

1.3.2 TPCompiladores.java

Classe que possui a main(), e que chama a classe Lexer.

1.3.3 Tag.java

Classe que define as constantes para os tokens, nela encontra-se todas as tags e seus respectivos valores instanciados.

1.3.4 Token.java

Representa um token genérico. Contém a constante que representa o token.

1.3.5 Word.java

Representa um token de palavras reservadas, identificadores e tokens compostos como `&&` e !=.

1.3.6 Tabela de Símbolos Utilizada

Tabela 1.1: Tabela de Símbolos

Tag	**Valor**
START	256
EXIT	257
END	258
ELSE	259
IF	300
THEN	301
SCAN	302
PRINT	303
DO	304
WHILE	305
OR	306
AND	307
ID	334
INT	308
INT	308
FLOAT	309
STRING	310
CHAR	311
NUM	312
LITERAL	313
EE	314
GT	315
GE	316
LT	317
LE	318
NOT	319
EQ	331
FALSE	320
TRUE	321
DOTCOMMA	322
OPENPARENT	323
CLOSEPARENT	324
ADD	325
SUB	326
MULT	327
DIV	328
COM	332
DOT	333
COMMENT	330
ERRO	400
FIMARQV	65535

1.4 Testes

Para testar a implementação do analisador léxico feita, foi proposto 5 arquivos de teste, disponibilizados pela professora, contendo código da gramática proposta e foi pedido que se criasse mais um arquivo com um código feito pelo grupo. Os resultados para cada um dos arquivos estão disponibilizado a seguir.

1.4.1 Teste 1

Tabela 1.2: Teste 1

Dados Arquivo	**Saída**	
start	Tag: 256	start
int a, b;	Tag: 308	int
int result;	Tag: 334	a
float a,x,total;	Tag: 332	,
a = 2;	Tag: 334	b
x = .1;	Tag: 322	;
scan (b);	Tag: 308	int
scan (y)	Tag: 334	result
result = (a*b ++ 1) / 2;	Tag: 322	;
print "Resultado: ";	Tag: 334	float
print (result);	Tag: 334	a
print ("Total: ");	Tag: 332	,
total = y / x;	Tag: 334	x
print ("Total: ";	Tag: 332	,
print (total);	Tag: 334	total
exit	Tag: 322	;
	Tag: 334	a
	Tag: 331	=
	Tag: 312	2
	Tag: 322	;
	Tag: 334	x
	Tag: 331	=
	Tag: 400	Lexema nao reconhecido - Linha: 6
	Tag: 312	1
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	b
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	y
	Tag: 324	)
	Tag: 334	result

Continua na próxima página

Tabela 1.2 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 331 =
	Tag: 323 (
	Tag: 334 a
	Tag: 327 *
	Tag: 334 b
	Tag: 325 +
	Tag: 325 +
	Tag: 312 1
	Tag: 324)
	Tag: 328 /
	Tag: 312 2
	Tag: 322 ;
	Tag: 303 print
	Tag: 310 "Resultado: "
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 334 result
	Tag: 324)
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 "Total: "
	Tag: 324)
	Tag: 322 ;
	Tag: 334 total
	Tag: 331 =
	Tag: 334 y
	Tag: 328 /
	Tag: 334 x
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 "Total: "
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 334 total
	Tag: 324)
	Tag: 322 ;
	Tag: 257 exit

1.4.2 Teste 1 Correto

O erro exibido na linha 6 era um erro de valor decimal em x=.1;

Tabela 1.3: Teste 1 Correto

Dados Arquivo	Saída	
start	Tag: 256	start
int a, b;	Tag: 308	int
int result;	Tag: 334	a
float a,x,total;	Tag: 332	,
a = 2;	Tag: 334	b
x = 1;	Tag: 322	;
scan (b);	Tag: 308	int
scan (y)	Tag: 334	result
result = (a*b ++ 1) / 2;	Tag: 322	;
print "Resultado: ";	Tag: 334	float
print (result);	Tag: 334	a
print ("Total: ");	Tag: 332	,
total = y / x;	Tag: 334	x
print ("Total: ";	Tag: 332	,
print (total);	Tag: 334	total
exit	Tag: 322	;
	Tag: 334	a
	Tag: 331	=
	Tag: 312	2
	Tag: 322	;
	Tag: 334	x
	Tag: 331	=
	Tag: 312	1
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	b
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	y
	Tag: 324	)
	Tag: 334	result
	Tag: 331	=
	Tag: 323	(
	Tag: 334	a
	Tag: 327	*
	Tag: 334	b
	Tag: 325	+
	Tag: 325	+

Continua na próxima página

Tabela 1.3 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 312	1
	Tag: 324	)
	Tag: 328	/
	Tag: 312	2
	Tag: 322	;
	Tag: 303	print
	Tag: 310	"Resultado: "
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 334	result
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	"Total: "
	Tag: 324	)
	Tag: 322	;
	Tag: 334	total
	Tag: 331	=
	Tag: 334	y
	Tag: 328	/
	Tag: 334	x
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	"Total: "
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 334	total
	Tag: 324	)
	Tag: 322	;
	Tag: 257	exit

1.4.3 Teste 2

Tabela 1.4: Teste 2

Dados Arquivo	Saída	
start	Tag: 256	start
int: a, c;	Tag: 308	int
float d, _e;	Tag: 334	a
		Continua na próxima página

Tabela 1.4 – continuação da página anterior

Dados Arquivo	Saída	
a = 0; d = 3.5	Tag: 334	,
c = d / 1.2;	Tag: 334	c
Scan (a);	Tag: 322	;
Scan (c);	Tag: 334	float
b = a * a;	Tag: 334	d
c = b + a * (1 + a*c);	Tag: 332	,
print (”Resultado: ”);	Tag: 400	Lexema nao reconhecido - Linha: 3
print c;	Tag: 334	e
d = 34.2	Tag: 322	;
e = val + 2.2;	Tag: 334	a
print (”E: ”);	Tag: 331	=
print (e);	Tag: 312	0
a = b + c + d)/2;	Tag: 322	;
	Tag: 334	d
	Tag: 331	=
	Tag: 312	3
	Tag: 400	Lexema nao reconhecido - Linha: 4
	Tag: 312	5
	Tag: 334	c
	Tag: 331	=
	Tag: 334	d
	Tag: 328	/
	Tag: 312	1
	Tag: 400	Lexema nao reconhecido - Linha: 5
	Tag: 312	2
	Tag: 322	;
	Tag: 334	Scan
	Tag: 323	(
	Tag: 334	a
	Tag: 324	)
	Tag: 322	;
	Tag: 334	Scan
	Tag: 323	(
	Tag: 334	c
	Tag: 324	)
	Tag: 322	;
	Tag: 334	b
	Tag: 331	=
	Tag: 334	a
	Tag: 327	*
	Tag: 334	a
	Tag: 322	;
	Tag: 334	c
	Tag: 331	=
	Tag: 334	b

Continua na próxima página

Tabela 1.4 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 325 +
	Tag: 334 a
	Tag: 327 *
	Tag: 323 (
	Tag: 312 1
	Tag: 325 +
	Tag: 334 a
	Tag: 327 *
	Tag: 334 c
	Tag: 324)
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 "Resultado: "
	Tag: 324)
	Tag: 322 ;
	Tag: 303 print
	Tag: 334 c
	Tag: 322 ;
	Tag: 334 d
	Tag: 331 =
	Tag: 312 34
	Tag: 400 Lexema nao reconhecido - Linha: 12
	Tag: 312 2
	Tag: 334 e
	Tag: 331 =
	Tag: 334 val
	Tag: 325 +
	Tag: 312 2
	Tag: 400 Lexema nao reconhecido - Linha: 13
	Tag: 312 2
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 "E: "
	Tag: 324)
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 334 e
	Tag: 324)
	Tag: 322 ;
	Tag: 334 a
	Tag: 331 =
	Tag: 334 b

Continua na próxima página

Tabela 1.4 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 325	+
	Tag: 334	c
	Tag: 325	+
	Tag: 334	d
	Tag: 324	)
	Tag: 328	/
	Tag: 312	2
	Tag: 322	;

1.4.4 Teste 2 Correto

Os erros apresentados foram erros de decimal e presença do caracter ":"que não foi especificado na linguagem.

Tabela 1.5: Teste 2 Correto

Dados Arquivo	Saída	
start	Tag: 256	start
int a, c;	Tag: 308	int
float d, e;	Tag: 334	a
a = 0; d = 3;	Tag: 332	,
c = d / 1;	Tag: 334	c
Scan (a);	Tag: 322	;
Scan (c);	Tag: 334	float
b = a * a;	Tag: 334	d
c = b + a * (1 + a*c);	Tag: 332	,
print ("Resultado: ");	Tag: 334	e
print c;	Tag: 322	;
d = 34	Tag: 334	a
e = val + 2;	Tag: 331	=
print ("E: ");	Tag: 312	0
print (e);	Tag: 322	;
a = b + c + d)/2;	Tag: 334	d
	Tag: 331	=
	Tag: 312	3
	Tag: 334	c
	Tag: 322	;
	Tag: 331	=
	Tag: 334	d
	Tag: 328	/
	Tag: 312	1
	Tag: 322	;
	Tag: 334	Scan

Continua na próxima página

Tabela 1.5 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 323 (
	Tag: 334 a
	Tag: 324)
	Tag: 322 ;
	Tag: 334 Scan
	Tag: 323 (
	Tag: 334 c
	Tag: 324)
	Tag: 322 ;
	Tag: 334 b
	Tag: 331 =
	Tag: 334 a
	Tag: 327 *
	Tag: 334 a
	Tag: 322 ;
	Tag: 334 c
	Tag: 331 =
	Tag: 334 b
	Tag: 325 +
	Tag: 334 a
	Tag: 327 *
	Tag: 323 (
	Tag: 312 1
	Tag: 325 +
	Tag: 334 a
	Tag: 327 *
	Tag: 334 c
	Tag: 324)
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 "Resultado: "
	Tag: 324)
	Tag: 322 ;
	Tag: 303 print
	Tag: 334 c
	Tag: 322 ;
	Tag: 334 d
	Tag: 331 =
	Tag: 312 34
	Tag: 334 e
	Tag: 331 =
	Tag: 334 val
	Tag: 325 +
	Tag: 312 2

Continua na próxima página

Tabela 1.5 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	”E: ”
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 334	e
	Tag: 324	)
	Tag: 322	;
	Tag: 334	a
	Tag: 331	=
	Tag: 334	b
	Tag: 325	+
	Tag: 334	c
	Tag: 325	+
	Tag: 334	d
	Tag: 324	)
	Tag: 328	/
	Tag: 312	2
	Tag: 322	;

1.4.5 Teste3

Tabela 1.6: Teste 3

Dados Arquivo	Saída	
int pontuacao, pontuacaoMaxina,		
disponibilidade;	Tag: 308	int
string pontuacaoMinima;	Tag: 334	pontuacao
disponibilidade = “Sim”;	Tag: 332	,
pontuacaoMinima = 50;	Tag: 334	pontuacaoMaxina
pontuacaoMaxima = 100;	Tag: 332	,
{ Entrada de dados	Tag: 334	disponibilidade
Verifica aprovação de candidatos	Tag: 322	;
do	Tag: 310	string
print(”Pontuacao Candidato: ”);	Tag: 334	pontuacaoMinima
scan(pontuacao);	Tag: 322	;
print(”Disp. Candidato: ”);	Tag: 334	disponibilidade
scan(disponibilidade);	Tag: 331	=
if ((pontuação > pontMin) and	Tag: 400	Lexema nao reconhecido-Linha:2
(disponibilidade==“Sim”) then	Tag: 334	Sim

Continua na próxima página

Tabela 1.6 – continuação da página anterior

Dados Arquivo	Saída	
out(”Candidato aprovado”);	Tag: 400	Lexema nao reconhecido-Linha:2
else	Tag: 322	;
out(”Candidato reprovado”)	Tag: 334	pontMin
end	Tag: 331	=
while (pontuação >= 0)end	Tag: 312	50
exit	Tag: 322	;
	Tag: 334	pontuacaoMaxima
	Tag: 331	=
	Tag: 312	100
	Tag: 322	;
	Tag: 400	Lexema nao reconhecido-Linha:14

1.4.6 Teste 3 Correto

Tabela 1.7: Teste 3 Correto

Dados Arquivo	Saída	
int pontuacao, pontuacaoMaxina,	Tag: 308	int
disponibilidade;	Tag: 334	pontuacao
string pontuacaoMinima;	Tag: 332	,
disponibilidade = ”Sim”;	Tag: 334	pontuacaoMaxina
pontuacaoMinima = 50;	Tag: 332	,
pontuacaoMaxima = 100;	Tag: 334	disponibilidade
{ Entrada de dados		
Verifica aprovação de candidatos }	Tag: 322	;
do	Tag: 310	string
print(”Pontuacao Candidato: ”);	Tag: 334	pontuacaoMinima
scan(pontuacao);	Tag: 322	;
print(”Disp. Candidato: ”);	Tag: 334	disponibilidade
scan(disponibilidade);	Tag: 331	=
if ((pontuacao > pontMin) and	Tag: 310	”Sim”
(disponibilidade==”Sim”))	Tag: 322	;
then out(”Candidato aprovado”);	Tag: 334	pontuacaoMinima
else	Tag: 331	=
out(”Candidato reprovado”);	Tag: 312	50
end	Tag: 322	;
while (pontuacao >= 0)end	Tag: 334	pontuacaoMaxima
exit	Tag: 331	=
	Tag: 312	100
	Tag: 322	;
	Tag: 330	{ Entrada de dados Verifica aprovação de candidatos}
	Tag: 304	do

Continua na próxima página

Tabela 1.7 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 303 print Tag: 323 (Tag: 310 "Pontuacao Candidato: " Tag: 324) Tag: 322 ; Tag: 302 scan Tag: 323 (Tag: 334 pontuacao Tag: 324) Tag: 322 ; Tag: 303 print Tag: 323 (Tag: 310 "Disponibilidade Candidato: " Tag: 324) Tag: 322 ; Tag: 302 scan Tag: 323 (Tag: 334 disponibilidade Tag: 324) Tag: 322 ; Tag: 300 if Tag: 323 (Tag: 323 (Tag: 334 pontuacao Tag: 315 >> Tag: 334 pontuacaoMinima Tag: 324) Tag: 307 and Tag: 323 (Tag: 334 disponibilidade Tag: 314 == Tag: 310 "Sim" Tag: 324) Tag: 324) Tag: 301 then Tag: 334 out Tag: 323 (Tag: 310 "Candidato aprovado" Tag: 324) Tag: 322 ; Tag: 259 else Tag: 334 out Tag: 323 (Tag: 310 "Candidato reprovado" Tag: 324)
	Continua na próxima página

Tabela 1.7 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 322	;
	Tag: 334	end
	Tag: 305	while
	Tag: 323	(
	Tag: 334	pontuacao
	Tag: 316	>=
	Tag: 312	0
	Tag: 324	)
	Tag: 334	end
	Tag: 257	exit

1.4.7 Teste4

Tabela 1.8: Teste 4

Dados Arquivo	Saída	
start	Tag: 256	start
int: a, aux\$, b;	Tag: 308	int
string nome, sobrenome, msg;	Tag: 400	Lexema nao reconhecido - Linha: 2
print(Nome:);	Tag: 334	a
scan (nome);	Tag: 332	,
print("Sobrenome: ");	Tag: 334	aux
scan (sobrenome);	Tag: 400	Lexema nao reconhecido - Linha: 2
msg = "Ola, "+ nome + " "+	Tag: 332	,
sobrenome + "!";	Tag: 334	b
msg = msg + 1;	Tag: 322	;
print (msg);	Tag: 310	string
scan (a);	Tag: 334	nome
scan(b);	Tag: 332	,
if (a>b) then	Tag: 334	sobrenome
aux = b;	Tag: 332	,
b = a;	Tag: 334	msg
a = aux;	Tag: 322	;
end;	Tag: 303	print
print ("Apos a troca: ");	Tag: 323	(
out(a);	Tag: 334	Nome
out(b)	Tag: 400	Lexema nao reconhecido - Linha: 4
exit	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	nome
	Tag: 324	)

Continua na próxima página

Tabela 1.8 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 "Sobrenome: "
	Tag: 324)
	Tag: 322 ;
	Tag: 302 scan
	Tag: 323 (
	Tag: 334 sobrenome
	Tag: 324)
	Tag: 322 ;
	Tag: 334 msg
	Tag: 331 =
	Tag: 310 "Ola, "
	Tag: 325 +
	Tag: 334 nome
	Tag: 325 +
	Tag: 310 " "
	Tag: 325 +
	Tag: 334 sobrenome
	Tag: 325 +
	Tag: 310 "!"
	Tag: 322 ;
	Tag: 334 msg
	Tag: 331 =
	Tag: 334 msg
	Tag: 325 +
	Tag: 312 1
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 334 msg
	Tag: 324)
	Tag: 322 ;
	Tag: 302 scan
	Tag: 323 (
	Tag: 334 a
	Tag: 324)
	Tag: 322 ;
	Tag: 302 scan
	Tag: 323 (
	Tag: 334 b
	Tag: 324)
	Tag: 322 ;
	Tag: 300 if

Continua na próxima página

Tabela 1.8 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 323 (
	Tag: 334 a
	Tag: 315 >
	Tag: 334 b
	Tag: 324)
	Tag: 301 then
	Tag: 334 aux
	Tag: 331 =
	Tag: 334 b
	Tag: 322 ;
	Tag: 334 b
	Tag: 331 =
	Tag: 334 a
	Tag: 322 ;
	Tag: 334 a
	Tag: 331 =
	Tag: 334 aux
	Tag: 322 ;
	Tag: 334 end
	Tag: 322 ;
	Tag: 303 print
	Tag: 323 (
	Tag: 310 ”Apos a troca: ”
	Tag: 324)
	Tag: 322 ;
	Tag: 334 out
	Tag: 323 (
	Tag: 334 a
	Tag: 324)
	Tag: 322 ;
	Tag: 334 out
	Tag: 323 (
	Tag: 334 b
	Tag: 324)
	Tag: 257 exit

1.4.8 Teste 4 Correto

Tabela 1.9: Teste 4 Correto

Dados Arquivo	Saída
start	Tag: 256 start
int a, aux, b;	Tag: 308 int
Continua na próxima página	

Tabela 1.9 – continuação da página anterior

Dados Arquivo	Saída	
string nome, sobrenome, msg;	Tag: 334	a
print(Nome);	Tag: 332	,
scan (nome);	Tag: 334	aux
print("Sobrenome: ");	Tag: 332	,
scan (sobrenome);	Tag: 334	b
msg = "Ola, "+ nome + " "+	Tag: 322	;
sobrenome + "!";	Tag: 310	string
msg = msg + 1;	Tag: 334	nome
print (msg);	Tag: 332	,
scan (a);	Tag: 334	sobrenome
scan(b);	Tag: 332	,
if (a>b) then	Tag: 334	msg
aux = b;	Tag: 322	;
b = a;	Tag: 303	print
a = aux;	Tag: 323	(
end;	Tag: 334	Nome
print ("Apos a troca: ");	Tag: 324	)
out(a);	Tag: 322	;
out(b);	Tag: 302	scan
exit	Tag: 323	(
	Tag: 334	nome
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	"Sobrenome: "
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	sobrenome
	Tag: 324	)
	Tag: 322	;
	Tag: 334	msg
	Tag: 331	=
	Tag: 310	"Ola, "
	Tag: 325	+
	Tag: 334	nome
	Tag: 325	+
	Tag: 310	" "
	Tag: 325	+
	Tag: 334	sobrenome
	Tag: 325	+
	Tag: 310	"!"
	Tag: 322	;

Continua na próxima página

Tabela 1.9 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 334 msg Tag: 331 = Tag: 334 msg Tag: 325 + Tag: 312 1 Tag: 322 ; Tag: 303 print Tag: 323 (Tag: 334 msg Tag: 324) Tag: 322 ; Tag: 302 scan Tag: 323 (Tag: 334 a Tag: 324) Tag: 322 ; Tag: 302 scan Tag: 323 (Tag: 334 b Tag: 324) Tag: 322 ; Tag: 300 if Tag: 323 (Tag: 334 a Tag: 315 >> Tag: 334 b Tag: 324) Tag: 301 then Tag: 334 aux Tag: 331 = Tag: 334 b Tag: 322 ; Tag: 334 b Tag: 331 = Tag: 334 a Tag: 322 ; Tag: 334 a Tag: 331 = Tag: 334 aux Tag: 322 ; Tag: 334 end Tag: 322 ; Tag: 303 print Tag: 323 (Tag: 310 ”Apos a troca: ”
Continua na próxima página	

Tabela 1.9 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 324	)
	Tag: 322	;
	Tag: 334	out
	Tag: 323	(
	Tag: 334	a
	Tag: 324	)
	Tag: 322	;
	Tag: 334	out
	Tag: 323	(
	Tag: 334	b
	Tag: 324	)
	Tag: 322	;
	Tag: 257	exit

1.4.9 Teste 5

Tabela 1.10: Teste 5

Dados Arquivo	Saída	
start	Tag: 256	start
int a, b, c, maior, outro;	Tag: 308	int
do	Tag: 334	a
print(”A”);	Tag: 332	,
scan(a);	Tag: 334	b
print(”B”);	Tag: 332	,
scan(b);	Tag: 334	c
print(”C”);	Tag: 332	,
scan(c);	Tag: 334	maior
{Realizacao do teste}	Tag: 332	,
if ((a > b) and (a>c))	Tag: 334	outro
maior = a	Tag: 322	;
else	Tag: 304	do
if (b > c) then	Tag: 303	print
maior = b;	Tag: 323	(
else	Tag: 310	”A”
maior = c;	Tag: 324	)
end	Tag: 322	;
end	Tag: 302	scan
print(”Maior valor:” ”);	Tag: 323	(
print (maior);	Tag: 334	a
print (”Outro? ”);	Tag: 324	)
scan(outro);	Tag: 322	;
while (outro >= 0)	Tag: 303	print

Continua na próxima página

Tabela 1.10 – continuação da página anterior

Dados Arquivo	Saída	
exit	Tag: 323	(
	Tag: 310	"B"
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	b
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	"C"
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	c
	Tag: 324	)
	Tag: 322	;
	Tag: 330	{Realizacao do teste}
	Tag: 300	if
	Tag: 323	(
	Tag: 323	(
	Tag: 334	a
	Tag: 315	>
	Tag: 334	b
	Tag: 324	)
	Tag: 307	and
	Tag: 323	(
	Tag: 334	a
	Tag: 315	>
	Tag: 334	c
	Tag: 324	)
	Tag: 324	)
	Tag: 334	maior
	Tag: 331	=
	Tag: 334	a
	Tag: 259	else
	Tag: 300	if
	Tag: 323	(
	Tag: 334	b
	Tag: 315	>
	Tag: 334	c
	Tag: 324	)
	Tag: 301	then

Continua na próxima página

Tabela 1.10 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 334	maior
	Tag: 331	=
	Tag: 334	b
	Tag: 322	;
	Tag: 259	else
	Tag: 334	maior
	Tag: 331	=
	Tag: 334	c
	Tag: 322	;
	Tag: 334	end
	Tag: 334	end
	Tag: 303	print
	Tag: 323	(
	Tag: 310	”Maior valor:”
	Tag: 400	Lexema nao reconhecido - Linha: 20
	Tag: 303	print
	Tag: 323	(
	Tag: 334	maior
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	”Outro? ”
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	outro
	Tag: 324	)
	Tag: 322	;
	Tag: 305	while
	Tag: 323	(
	Tag: 334	outro
	Tag: 316	>=
	Tag: 312	0
	Tag: 324	)
	Tag: 257	exit

1.4.10 Teste 5 Correto

Tabela 1.11: Teste 5 Correto

Dados Arquivo	Saída	
start	Tag: 256	start
int a, b, c, maior, outro;	Tag: 308	int
do	Tag: 334	a
print(”A”);	Tag: 332	,
scan(a);	Tag: 334	b
print(”B”);	Tag: 332	,
scan(b);	Tag: 334	c
print(”C”);	Tag: 332	,
scan(c);	Tag: 334	maior
{Realizacao do teste}	Tag: 332	,
if ((a>b) and (a>c))	Tag: 334	outro
maior = a	Tag: 322	;
else	Tag: 304	do
if (b>c) then	Tag: 303	print
maior = b;	Tag: 323	(
else	Tag: 310	”A”
maior = c;	Tag: 324	)
end end	Tag: 322	;
print(”Maior valor:”);	Tag: 302	scan
print (maior);	Tag: 323	(
print (”Outro? ”);	Tag: 334	a
scan(outro);	Tag: 324	)
while (outro ¿= 0)	Tag: 322	;
exit	Tag: 303	print
	Tag: 323	(
	Tag: 310	”B”
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	b
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	”C”
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	c
	Tag: 324	)
	Tag: 322	;
	Tag: 330	{Realizacao do teste}

Continua na próxima página

Tabela 1.11 – continuação da página anterior

Dados Arquivo	Saída
	Tag: 300 if Tag: 323 (Tag: 323 (Tag: 334 a Tag: 315 >> Tag: 334 b Tag: 324) Tag: 307 and Tag: 323 (Tag: 334 a Tag: 315 >> Tag: 334 c Tag: 324) Tag: 324) Tag: 334 maior Tag: 331 = Tag: 334 a Tag: 259 else Tag: 300 if Tag: 323 (Tag: 334 b Tag: 315 >> Tag: 334 c Tag: 324) Tag: 301 then Tag: 334 maior Tag: 331 = Tag: 334 b Tag: 322 ; Tag: 259 else Tag: 334 maior Tag: 331 = Tag: 334 c Tag: 322 ; Tag: 334 end Tag: 334 end Tag: 303 print Tag: 323 (Tag: 310 ”Maior valor:” Tag: 324) Tag: 322 ; Tag: 303 print Tag: 323 (Tag: 334 maior Tag: 324)
	Continua na próxima página

Tabela 1.11 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 310	”Outro? ”
	Tag: 324	)
	Tag: 322	;
	Tag: 302	scan
	Tag: 323	(
	Tag: 334	outro
	Tag: 324	)
	Tag: 322	;
	Tag: 305	while
	Tag: 323	(
	Tag: 334	outro
	Tag: 316	>=
	Tag: 312	0
	Tag: 324	)
	Tag: 257	exit

1.4.11 Teste 6

Tabela 1.12: Teste 6

Dados Arquivo	Saída	
start	Tag: 256	start
int fatorial;	Tag: 308	int
int f=5;	Tag: 334	fatorial
print(”O fatorial de ”);	Tag: 322	;
print(f);	Tag: 308	int
do	Tag: 334	f
fatorial = f * (f-1);	Tag: 331	=
while(f-1 ¿ 0) end	Tag: 312	5
print(”e: ”);	Tag: 322	;
print(fatorial);	Tag: 303	print
exit	Tag: 323	(
	Tag: 310	”O fatorial de ”
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 334	f
	Tag: 324	)
	Tag: 322	;

Continua na próxima página

Tabela 1.12 – continuação da página anterior

Dados Arquivo	Saída	
	Tag: 304	do
	Tag: 334	fatorial
	Tag: 331	=
	Tag: 334	f
	Tag: 327	*
	Tag: 323	(
	Tag: 334	f
	Tag: 326	-
	Tag: 312	1
	Tag: 324	)
	Tag: 322	;
	Tag: 305	while
	Tag: 323	(
	Tag: 334	f
	Tag: 326	-
	Tag: 312	1
	Tag: 315	¿
	Tag: 312	0
	Tag: 324	)
	Tag: 334	end
	Tag: 303	print
	Tag: 323	(
	Tag: 310	”e: ”
	Tag: 324	)
	Tag: 322	;
	Tag: 303	print
	Tag: 323	(
	Tag: 334	fatorial
	Tag: 324	)
	Tag: 322	;
	Tag: 257	exit

2 Entrega Analisador Sintático

2.1 Introdução

Análise sintática (também conhecido pelo termo em inglês parsing) é o processo de analisar uma sequência de entrada (lida de um arquivo de computador ou do teclado, por exemplo) para determinar sua estrutura gramatical segundo uma determinada gramática formal. Essa análise faz parte de um compilador, junto com a análise léxica e análise semântica. A análise sintática, ou análise gramatical é o processo de se determinar se uma cadeia de símbolos léxicos pode ser gerada por uma gramática.

A análise sintática transforma um texto na entrada em uma estrutura de dados, em geral uma árvore, o que é conveniente para processamento posterior e captura a hierarquia implícita desta entrada. Através da análise léxica é obtido um grupo de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura.

Em termos práticos, pode também ser usada para decompor um texto em unidades estruturais para serem organizadas dentro de um bloco, por exemplo.

2.2 Forma de Uso do Compilador

O projeto continua sendo desenvolvido no Netbeans. Para isso, deve-se baixar o arquivo compac-tado, descompactar, abrir o projeto no Netbeans e rodar. Todos os testes estão na pasta raíz do projeto. Bastando que altere o nome que se encontra no arquivo TPCompiladores.java. Os arquivos foram nomiados de acordo com a especificação do trabalho. Sendo eles:"teste1.txt", "teste2.txt", "teste3.txt", "teste4.txt", "teste5.txt"e "teste6.txt".

2.3 Descrição da Abordagem Utilizada

Para a implementação do analisador sintático foi necessário a implementação de mais três classes ao projeto.

2.3.1 Int.java

Classe que herda de token. É uma especialização de Token para o caso especial em que o token é do tipo INT.

2.3.2 Float.java

Classe que herda de token. É uma especialização de Token para o caso especial em que o token é do tipo FLOAT.

2.3.3 Sintax.java

Classe que implementa o analisador sintático. A classe tem como um de seus atributos o lex, que é do tipo Lexer para receber os tokens reconhecidos no codigo.

2.4 Testes

Para testar a implementação do analisador sintático feita, foi proposto 5 arquivos de teste, disponibilizados pela professora, contendo código da gramática proposta e foi pedido que se criasse mais um arquivo com um código feito pelo grupo. Os resultados para cada um dos arquivos estão disponibilizado a seguir.

2.4.1 Teste 1

Tabela 2.1: Teste 1

Dados Arquivo	**Saída**

start
int a, b;
int result;
float a,x,total;
a = 2;
x = .1;
scan (b);
scan (y)
result = (a*b ++ 1) / 2;
print "Resultado: ";
print (result);
print ("Total: ");
total = y / x;
print ("Total: ";
print (total);
exit
	program();
	EAT(start)
	delc_list();
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(a)
	EAT(,)
	ident_list();
	identifier();
	EAT(b)
	EAT(;)
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(result)
	EAT(;)
	delc_list();
	delc();
	type();
	EAT(float)
	ident_list();
	identifier();
	EAT(a)
	EAT(,)
	ident_list();

Continua na próxima página	

Tabela 2.1 – continuação da página anterior

Dados Arquivo	Saída

	identifier();
	EAT(x)
	EAT(,)
	ident_list();
	identifier();
	EAT(total)
	EAT(;)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(a)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(2)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(x)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	Erro - Token inesperado
	Erro - Token tag: 322
	Erro - Token tag: 257

2.4.2 Teste 1 - Correto

Tabela 2.2: Teste 1 - Correto

Dados Arquivo	Saída

start
int a, b;
int result;
float a, x, total;
	program();
	EAT(start)
	delc_list();
	delc();

Continua na próxima página	

Tabela 2.2 – continuação da página anterior

Dados Arquivo	Saída

a = 2;
b = 2;
x = 0.1;
scan (b);
scan (y);
result = (a * b + 1) / 2;
print (”Resultado: ”);
print (result);
print (”Total: ”);
total = y / x;
print (”Total: ”);
print (total);
exit
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(a)
	EAT(,)
	ident_list();
	identifier();
	EAT(b)
	EAT(;)
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(result)
	EAT(;)
	delc_list();
	delc();
	type();
	EAT(float)
	ident_list();
	identifier();
	EAT(a)
	EAT(,)
	ident_list();
	identifier();
	EAT(x)
	EAT(,)
	ident_list();
	identifier();
	EAT(total)
	EAT(;)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(a)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();

Continua na próxima página	

Tabela 2.2 – continuação da página anterior

Dados Arquivo	Saída

	EAT(2)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(b)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(2)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(x)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	float_const();
	EAT(0.1)
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(b)
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(y)
	EAT())
	EAT(;)
	stmt();

Continua na próxima página	

Tabela 2.2 – continuação da página anterior

Dados Arquivo	Saída

	assign_stmt();
	identifier();
	EAT(result)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	term2();
	mulop();
	EAT(*)
	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(1)
	EAT())
	term2();
	mulop();
	EAT(/)
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(2)
	EAT(;)

Continua na próxima página	

Tabela 2.2 – continuação da página anterior

Dados Arquivo	Saída

	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Resultado: ")
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(result)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Total: ")
	EAT())
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(total)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(y)
	term2();
	mulop();
	EAT(/)

Continua na próxima página	

Tabela 2.2 – continuação da página anterior

Dados Arquivo	Saída

	term();
	factor_a();
	factor();
	identifier();
	EAT(x)
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Total: ")
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(total)
	EAT())
	EAT(;)
	EAT(exit)

2.4.3 Teste 2

Tabela 2.3: Teste 2

Dados Arquivo	Saída
start	program(); EAT(start)
int: a, c;	delc_list();
float d, _e;	delc();
a = 0; d = 3.5	type();
c = d / 1.2;	EAT(int)
Scan (a);	ident_list();
Scan (c);	identifier();
b = a * a;	Erro - Token inesperado
Continua na próxima página	

Tabela 2.3 – continuação da página anterior

Dados Arquivo	Saída
c = b + a * (1 + a*c);	Erro - Token tag: 322
print (”Resultado: ”);	Erro - Token tag: 322
print c;	stmt_list();
d = 34.2	stmt();
e = val + 2.2;	Erro - Token tag: 257
print (”E: ”);	
print (e);	
a = b + c + d)/2;	

2.4.4 Teste 2 - Correto

Tabela 2.4: Teste 2 - Correto

Dados Arquivo	Saída
start	program();
int a, c;	EAT(start)
float d, e;	delc_list();
a = 0;	delc();
d = 35;	type();
c = d / 12;	EAT(int)
scan(a);	ident_lis1t();
scan(c);	identifier();
b = a * a;	EAT(a)
c = b + a * (1 + a * c);	EAT(,)
print (”Resultado: ”);	ident_list();
print(c);	identifier();
d = 34;	EAT(c)
e = d + 2;	EAT(;)
print(”E: ”);	delc();
print(e);	type();
a = (b + c + d)/2;	EAT(float)
exit	ident_list();
	identifier();
	EAT(d)
	EAT(,)
	ident_list();
	identifier();
	EAT(e)
	EAT(;)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();

Continua na próxima página

Tabela 2.4 – continuação da página anterior

Dados Arquivo	Saída

	EAT(a)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(0)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(d)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(35)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(c)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(d)
	term();
	mulop();
	EAT(/)
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(12)
	EAT(;)
	stmt();

Continua na próxima página	

Tabela 2.4 – continuação da página anterior

Dados Arquivo	Saída

	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(a)
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(c)
	EAT())
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(b)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	term();
	mulop();
	EAT(*)
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(c)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();

Continua na próxima página	

Tabela 2.4 – continuação da página anterior

Dados Arquivo	Saída

	EAT(b)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	term();
	mulop();
	EAT(*)
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(1)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	term();
	mulop();
	EAT(*)
	term();
	factor_a();
	factor();
	identifier();
	EAT(c)
	EAT())
	EAT(;)
	stmt();

Continua na próxima página	

Tabela 2.4 – continuação da página anterior

Dados Arquivo	Saída

	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Resultado: ")
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(c)
	EAT())
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(d)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(34)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(e)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(d)

Continua na próxima página	

Tabela 2.4 – continuação da página anterior

Dados Arquivo	Saída

	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(2)
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("E: ")
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(e)
	EAT())
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(a)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();

Continua na próxima página	

Tabela 2.4 – continuação da página anterior

Dados Arquivo	Saída

	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(c)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(d)
	EAT())
	term();
	mulop();
	EAT(/)
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(2)
	EAT(;)
	EAT(exit)

2.4.5 Teste 3

Tabela 2.5: Teste 3

Dados Arquivo	Saída
int pontuacao, pontuacaoMaxina, disponibilidade;	Fatal Error: Esperava-se token inicializador.
Continua na próxima página	

Tabela 2.5 – continuação da página anterior

Dados Arquivo	Saída

string pontuacaoMinima;
disponibilidade = “Sim”;
pontuacaoMinima = 50;
pontuacaoMaxima = 100;
{ Entrada de dados
Verifica aprovação de candidatos
do
print(”Pontuacao Candidato: ”);
scan(pontuacao);
print(”Disp. Candidato: ”);
scan(disponibilidade);
if ((pontuação > pontMin) and
(disponibilidade==“Sim”) then
out(”Candidato aprovado”);
else
out(”Candidato reprovado”)
end
while (pontuação >= 0)end
exit	

2.4.6 Teste 3 - correto

Tabela 2.6: Teste 3 - correto

Dados Arquivo	Saída

start
int pontuacao, pontuacaoMaxina;
string pontuacaoMinima, disponibilidade;
disponibilidade = ”Sim”;
pontuacaoMinima = 50;
pontuacaoMaxima = 100;
Entrada de dados
Verifica aprovacao de candidatos
do
print(”Pontuacao Candidato: ”);
scan(pontuacao);
print(”Disponibilidade Candidato: ”);
scan(disponibilidade);
if ((pontuacao ¿ pontuacaoMinima) and
(disponibilidade==”Sim”)) then
print(”Candidato aprovado”);
else
print(”Candidato reprovado”);
	program();
	EAT(start)
	delc_list();
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(pontuacao)
	EAT(,)
	ident_list();
	identifier();
	EAT(pontuacaoMaxina)
	EAT(;)
	delc();
	type();
	EAT(string)
	ident_list();

Continua na próxima página	

Tabela 2.6 – continuação da página anterior

Dados Arquivo	Saída

end
while (pontuacao ¿= 0)end
exit
	identifier();
	EAT(pontuacaoMinima)
	EAT(,)
	ident_list();
	identifier();
	EAT(disponibilidade)
	EAT(;)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(disponibilidade)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	literal();
	EAT("Sim")
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(pontuacaoMinima)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(50)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(pontuacaoMaxima)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();

Continua na próxima página

Tabela 2.6 – continuação da página anterior

Dados Arquivo	Saída

	EAT(100)
	EAT(;)
	comment();
	EAT({ Entrada de dados
	Verifica aprovacao de candidatos })
	stmt();
	while_stmt();
	EAT(do)
	stmt_list();
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Pontuacao Candidato: ")
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(pontuacao)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Disponibilidade Candidato: ")
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(disponibilidade)
	EAT())
	EAT(;)
	stmt();
	if_stmt();

	Continua na próxima página

Tabela 2.6 – continuação da página anterior

Dados Arquivo	Saída

	EAT(if)
	condition();
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(pontuacao)
	relop();
	EAT(¿)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(pontuacaoMinima)
	EAT())
	term2();
	mulop();
	EAT(and)
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(disponibilidade)
	relop();

Continua na próxima página	

Tabela 2.6 – continuação da página anterior

Dados Arquivo	Saída

	EAT(==)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	literal();
	EAT("Sim")
	EAT())
	EAT())
	EAT(then)
	stmt_list();
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Candidato aprovado")
	EAT())
	EAT(;)
	EAT(else)
	stmt_list();
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Candidato reprovado")
	EAT())
	EAT(;)
	EAT(end)
	stmt_sufix();
	EAT(while)
	EAT(()
	condition();
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(pontuacao)
	relop();

	Continua na próxima página

Tabela 2.6 – continuação da página anterior

Dados Arquivo	Saída

	EAT(¿=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(0)
	EAT())
	EAT(end)
	EAT(exit)

2.4.7 Teste 4

Tabela 2.7: Teste 4

Dados Arquivo	Saída

start
int: a, aux\$, b;
string nome, sobrenome, msg;
print(Nome:);
scan (nome);
print(”Sobrenome: ”);
scan (sobrenome);
msg = ”Ola, ”+ nome + ” ”+
sobrenome + ”!”;
msg = msg + 1;
print (msg);
scan (a);
scan(b);
if (a>b) then
aux = b;
b = a;
a = aux;
end;
print (”Apos a troca: ”);
out(a);
out(b)
exit
	program();
	EAT(start)
	delc_list();
	delc();
	type();
	ident_list();
	identifier();
	Erro - Token inesperado
	Erro - Token tag: 322
	stmt_list();
	stmt();
	Erro - Token tag: 257

2.4.8 Teste 4 - correto

Tabela 2.8: Teste 4 - correto

Dados Arquivo	Saída

start
int a, aux, b;
string nome, sobrenome, msg;
print(”Nome: ”);
scan (nome);
print(”Sobrenome: ”);
scan (sobrenome);
msg = ”Ola, ”+ nome + ” ”+
sobrenome + ”!”;
msg = msg + 1;
print (msg);
scan (a);
scan(b);
if (a>b) then
aux = b;
b = a;
a = aux;
end;
print (”Apos a troca: ”);
exit
	program();
	EAT(start)
	delc_list();
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(a)
	EAT(,)
	ident_list();
	identifier();
	EAT(aux)
	EAT(,)
	ident_list();
	identifier();
	EAT(b)
	EAT(;)
	delc();
	type();
	EAT(string)
	ident_list();
	identifier();
	EAT(nome)
	EAT(,)
	ident_list();
	identifier();
	EAT(sobrenome)
	EAT(,)
	ident_list();
	identifier();
	EAT(msg)
	EAT(;)
	stmt_list();
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT(”Nome: ”)
	EAT())

Continua na próxima página	

Tabela 2.8 – continuação da página anterior

Dados Arquivo	Saída

	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(nome)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Sobrenome: ")
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(sobrenome)
	EAT())
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(msg)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	literal();
	EAT("Ola, ")
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();

	Continua na próxima página

Tabela 2.8 – continuação da página anterior

Dados Arquivo	Saída

	identifier();
	EAT(nome)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	literal();
	EAT()
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(sobrenome)
	simple_expr2();
	addop();
	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	literal();
	EAT("!")
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(msg)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(msg)
	simple_expr2();
	addop();

	Continua na próxima página

Tabela 2.8 – continuação da página anterior

Dados Arquivo	Saída

	EAT(+)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(1)
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(msg)
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(a)
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(b)
	EAT())
	EAT(;)
	stmt();
	if_stmt();
	EAT(if)
	condition();
	expression();
	simple_expr();
	term();

	Continua na próxima página

Tabela 2.8 – continuação da página anterior

Dados Arquivo	Saída

	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	relop();
	EAT(¿)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	EAT())
	EAT(then)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(aux)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	EAT(;)
	stmt();
	assign_stmt();
	identifier();
	EAT(b)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	EAT(;)

	Continua na próxima página

Tabela 2.8 – continuação da página anterior

Dados Arquivo	Saída

	stmt();
	assign_stmt();
	identifier();
	EAT(a)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(aux)
	EAT(;)
	EAT(end)
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Apos a troca: ")
	EAT())
	EAT(;)
	EAT(exit)

2.4.9 Teste 5

Tabela 2.9: Teste 5

Dados Arquivo	Saída

start
int a, b, c, maior, outro;
do
print("A");
scan(a);
print("B");
scan(b);
print("C");
scan(c);
{Realizacao do teste}
if ((a > b) and (a>c))
maior = a
else
	program();
	Token:"start"
	EAT(start)
	delc-list();
	delc();
	type();
	Token:"int"
	EAT(int)
	ident-list();
	identifier();
	Token:"a"
	EAT(a)
	Token:",";

Continua na próxima página	

Tabela 2.9 – continuação da página anterior

Dados Arquivo	Saída
if (b > c) then	EAT(,)
maior = b;	ident-list();
else	identifier();
maior = c;	EAT(b)
end	Token:”,”
end	EAT(,)
print(”Maior valor:” ”);	ident-list();
print (maior);	identifier();
print (”Outro? ”);	Token:”c”
scan(outro);	EAT(c)
while (outro >= 0)	Token:”,”
exit	EAT(,)
	ident-list();
	identifier();
	Token:”maior”
	EAT(maior)
	Token:”,”
	EAT(,)
	ident-list();
	identifier();
	Token:”outro”
	EAT(outro)
	Token:”;”
	EAT(;)
	stmt-list();
	stmt();
	Token:”do”
	Erro - Token tag:257

2.4.10 Teste 5 - correto

Tabela 2.10: Teste 5 - correto

Dados Arquivo	Saída
start	program();
int a, b, c, maior, outro;	EAT(start)
do	delc_list();
print(”A”);	delc();
scan(a);	type();
print(”B”);	EAT(int)
scan(b);	ident_list();
print(”C”);	identifier();
scan(c);	EAT(a)
Continua na próxima página	

Tabela 2.10 – continuação da página anterior

Dados Arquivo	Saída
{Realizacao do teste}	EAT(,)
if ((a > b) and (a>c)) then	ident_list();
maior = a;	identifier();
else	EAT(b)
if (b > c) then	EAT(,)
maior = b;	ident_list();
else	identifier();
maior = c;	EAT(c)
end	EAT(,)
end	ident-list();
print("Maior valor:" ");	identifier();
print (maior);	EAT(maior)
print ("Outro? ");	EAT(,)
scan(outro);	ident_list();
while (outro >= 0)	identifier();
exit	EAT(outro)
	EAT(;)
	stmt_list();
	stmt();
	while_stmt();
	EAT(do)
	stmt_list();
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("A")
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(a)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();

Continua na próxima página

Tabela 2.10 – continuação da página anterior

Dados Arquivo	Saída

	EAT(”B”)
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(b)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT(”C”)
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(c)
	EAT())
	EAT(;)
	comment();
	EAT(Realizacao do teste)
	stmt();
	if_stmt();
	EAT(if)
	condition();
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();

Continua na próxima página	

Tabela 2.10 – continuação da página anterior

Dados Arquivo	Saída

	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	relop();
	EAT(>)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	EAT())
	term2();
	mulop();
	EAT(and)
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	relop();
	EAT(>)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(c)
	EAT())
	EAT())
	EAT(then)
	stmt_list();
	stmt();
	assign_stmt();

	Continua na próxima página

Tabela 2.10 – continuação da página anterior

Dados Arquivo	Saída

	identifier();
	EAT(maior)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(a)
	EAT(;)
	EAT(else)
	stmt_list();
	stmt();
	if_stmt();
	EAT(if)
	condition();
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	relop();
	EAT(>)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(c)
	EAT())
	EAT(then)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(maior)
	EAT(=)

Continua na próxima página	

Tabela 2.10 – continuação da página anterior

Dados Arquivo	Saída

	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(b)
	EAT(;)
	EAT(else)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(maior)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(c)
	EAT(;)
	EAT(end)
	EAT(end)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Maior valor:")
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(maior)
	EAT())
	EAT(;)

Continua na próxima página	

Tabela 2.10 – continuação da página anterior

Dados Arquivo	Saída

	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Outro? ")
	EAT())
	EAT(;)
	stmt();
	read_stmt();
	EAT(scan)
	EAT(()
	identifier();
	EAT(outro)
	EAT())
	EAT(;)
	stmt_sufix();
	EAT(while)
	EAT(()
	condition();
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(outro)
	relop();
	EAT>=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(0)
	EAT())
	EAT(end)
	EAT(exit)

2.4.11 Teste 6

Tabela 2.11: Teste 6

Dados Arquivo	**Saída**

start
int fatorial;
int f;
f=5;
print(”O fatorial de ”);
print(f);
do
fatorial = f * (f-1);
while(f-1 ¿ 0) end
print(”e: ”);
print(fatorial);
exit
	program();
	EAT(start)
	delc_list();
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(fatorial)
	EAT(;)
	delc();
	type();
	EAT(int)
	ident_list();
	identifier();
	EAT(f)
	EAT(;)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(f)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(5)
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT(”O fatorial de ”)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()

Continua na próxima página	

Tabela 2.11 – continuação da página anterior

Dados Arquivo	Saída

	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(f)
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("Ola")
	EAT())
	EAT(;)
	stmt();
	while_stmt();
	EAT(do)
	stmt_list();
	stmt();
	assign_stmt();
	identifier();
	EAT(fatorial)
	EAT(=)
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(f)
	term2();
	mulop();
	EAT(*)
	term();
	factor_a();
	factor();
	EAT(()
	expression();
	simple_expr();
	term();
	factor_a();
	factor();

	Continua na próxima página

Tabela 2.11 – continuação da página anterior

Dados Arquivo	Saída

	identifier();
	EAT(f)
	simple_expr2();
	addop();
	EAT(-)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(1)
	EAT())
	EAT(;)
	stmt_sufix();
	EAT(while)
	EAT(()
	condition();
	expression();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(f)
	simple_expr2();
	addop();
	EAT(-)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(1)
	relop();
	EAT(¿)
	simple_expr();
	term();
	factor_a();
	factor();
	constant();
	inter_const();
	EAT(0)
	EAT())

	Continua na próxima página

Tabela 2.11 – continuação da página anterior

Dados Arquivo	Saída

	EAT(end)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	literal();
	EAT("e: ")
	EAT())
	EAT(;)
	stmt();
	write_stmt();
	EAT(print)
	EAT(()
	writable();
	simple_expr();
	term();
	factor_a();
	factor();
	identifier();
	EAT(fatorial)
	EAT())
	EAT(;)
	EAT(exit)