CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO

IMPLEMENTAÇÃO COMPILADOR
PARTE I

ANALISADOR LÉXICO

Autor: Alan Ferreira Leite Santos
Marcos Junio da Silva Xavier
Samuel Filipe dos Santos
Orientador: Kecia Aline Marques Ferreira

19 de fevereiro de 2021, Belo Horizonte

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO

Alan Ferreira Leite Santos

Marcos Junio da Silva Xavier
Samuel Filipe dos Santos

IMPLEMENTAÇÃO COMPILADOR
PARTE I- ANALISADOR LÉXICO

Trabalho realizado em consonância com os princípios teóricos aprendidos durante o semestre a fim de criar, por etapas, um compilador para uma determinada linguagem proposta pela orientadora.
Orientador: Kecia Aline Marques Ferreira

19 de fevereiro de 2021, Belo Horizonte

RESUMO

Na disciplina de Compiladores, ofertada pelo CEFET-MG, iremos construir um compilador ao longo do semestre, este, dividido em etapas. Nesta primeira etapa, estamos entregando a implementação do analisador léxico e a Tabela de Símbolos em Java, conhecimentos obtidos com a disciplina orientados pela Dr Kecia Ferreira, para que seja possível, utilizamos como base os conceitos abordados pelo Aho, em seu livro : ”Compiladores Princípios ,Técnicas e Ferramentas”

Palavras Chaves: java,compilador

ABSTRACT

In the course of Compilers, offered by CEFET-MG, we will build a compiler throughout the semester, this one, divided into stages. In this first stage, we are delivering the Java implementation of the lexical analyzer and the Symbol Table, knowledge obtained with the discipline guided by Dr Kecia Ferreira, to make it possible, we use as a basis the concepts covered by Aho, in his book: ”Compilers Principles, Techniques and Tools”

Key-words: java,compiler

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIMBOLOS

$\backslash n$	Quebra de Linha
(	Abertura de Parenteses
)	Fechamento de parenteses
$*$	Operador de Multiplicação
$+$	Operador de Adição
$-$	Operador de Subtração
$/$	Operador de Divisão
$:=$	Operador de Atribuição
;	Indicador de Fim de Linha
$<$	Menor que
$<=$	Menor ou igual a
$<>$	Diferente
$>$	Maior que
$>=$	Maior ou igual a
%	Representa o inicio de um comentário

Sumário

1. INTRODUÇÃO

Um compilador é um programa de computador (ou um grupo de programas) que, a partir de um código fonte escrito em uma linguagem compilada, cria um programa semanticamente equivalente, porém escrito em outra linguagem, código objeto. 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. Atualmente, porém, são comuns compiladores que geram código para uma máquina virtual que é, depois, interpretada por um interpretador. Ele é chamado compilador por razões históricas; nos primeiros anos da programação automática, existiam programas que percorriam bibliotecas de sub-rotinas e as reunia, ou compilava, as sub-rotinas necessárias para executar uma determinada tarefa.

O nome ”compilador”é usado principalmente para os programas que traduzem o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (por exemplo, Assembly ou código de máquina). Contudo alguns autores citam exemplos de compiladores que traduzem para linguagens de alto nível como C. Para alguns autores um programa que faz uma tradução entre linguagens de alto nível é normalmente chamado um tradutor, filtro ou conversor de linguagem. Um programa que traduz uma linguagem de programação de baixo nível para uma linguagem de programação de alto nível é um descompilador. Um programa que faz uma tradução entre uma linguagem de montagem e o código de máquina é denominado montador (assembler). Um programa que faz uma tradução entre o código de máquina e uma linguagem de montagem é denominado desmontador (disassembler). Se o programa compilado pode ser executado em um computador cuja CPU ou sistema operacional é diferente daquele em que o compilador é executado, o compilador é conhecido como um compilador cruzado.

2. PALAVRAS RESERVADAS

Para que o compilador seja executado de forma correta, existem algumas regrinhas a serem seguidas, uma delas é a definição de palavras reservadas, essas, são palavras que não devem ser utilizadas em nenhum outro momento a não ser o que foi pre-determinado pelos programadores da linguagem. Como por exemplo: No caso deste compilador, existe a palavra stop, a qual significa que quando o compilador ler esse token, significa que o programa chegou ao final, logo, em nenhum outro local esse token deve ter outro significado a não ser o significado que já foi determinado. Não se pode atribuir uma outra função a uma palavra reservada.

Tabela 1 – Relação das Palavras Reservadas da Linguagem

Palavra Reservada	Significado
init	Inicio do Programa
stop	Indica a finalização do programa
is	Usado para determinador o tipo de uma variável
integer	Tipo Inteiro para variável
string	Determina a variável como tipo string
real	Determina a variável como tipo Real
if	Determina o inicio de um bloco de código sob uma condicional
begin	Deternia o inicio de uma condição do if
end	Determina o fim da condição do if
else	Determina a contraposição da condição do if
end else	Indica o Fim da condição do Else if
do	Determina a entrada de um laço
while	determina a condição para o laço Do
read	Prepara para ler e reservar a próxima variável a ser lida
write	Determina que ira imprimir na tela a literal que virá a seguir
not	Determina a negação do valor booleano de uma expressão
or	Determina a soma binaria entre dois valores
and	Determina a multiplicação binaria entre dois valores

Fonte: Documentação Fornecida pela Orientadora o qual pode ser consultado no diretório do projeto

3. FORMA DE UTILIZAÇÃO

Para que seja possivel a utilização deste programa, basta entrar no terminal e navegar até o diretório: ”TP_Compilador/src/”, e então, digitar o seguinte comando :

```
1    javac Main.java
2    java Main
```

Listing 3.1: Entrada terminal

Acima podemos ver duas linhas de comando, a primeira serve para que possa compilar a classe Main.java e então, é criado o executável e assim, podemos acessar a classe a partir da linha seguinte, java Main .
Definimos qual arquivo de teste que o programa deve compilar dentro da classe Main, no seguinte trecho de código:

```
1    public static void main(String[] args) {
2    ArrayList<Token> tokens = new ArrayList<Token> ();
3    Lexer L = null;
4    int line = -5;
5    try {
6      L = new Lexer("codigos_teste/Teste2.txt");
7
```

Listing 3.2: Indicando arquivo para teste

Na ultima linha do trecho do código destacado acima, mostra como determinamos qual programa nosso compilador irá testar. Veja que criamos uma pasta chamada testes para que seja incluídos somente os arquivos de teste do programa. Foram criados 10 arquivos de teste.

4. DOCUMENTAÇÃO

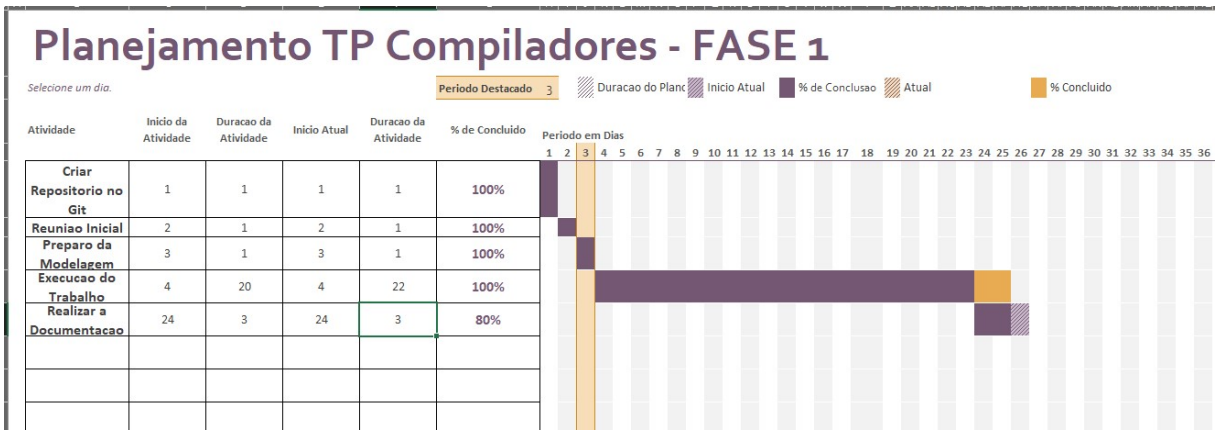
O Analisador Léxico é o processo de analisar a entrada de linhas de caracteres (tal como o código-fonte de um programa de computador) e produzir uma sequência de símbolos chamado ”símbolos léxicos”(lexical tokens), ou somente ”símbolos”(tokens), que podem ser manipulados mais facilmente por um parser (leitor de saída). Para tal procedimento, foi necessário a realização a partir do modelo de uma maquina de estados, processando os caracteres do arquivo fonte, montando lexemas e vinculando a um token especifico, verificando também a Tabela de Símbolos e respeitando a gramatica da linguagem. O Projeto foi desenvolvido em conjunto com as plataformas Visual Studio Code e Apache Netbeans, a principio todo o código seria criado somente ”a mão”criando as pastas e arquivos manualmente, porém, devido a algumas facilidades que o Netbeans nos oferece, o projeto foi alterado em alguns parâmetros para que seja possível abrir no Netbeans. Uma dessas facilidades é a opção de geração da Documentação do projeto, assim o fizemos e esta documentação gerada automaticamente pelo Netbeans se encontra no diretório: ”/TP_Compilador/Projeto/dist/javadoc/”. Para esta implementação foi acatado o modelo abordado pelo autor do livro base da disciplina: Aho .

5. PLANEJAMENTO

Para que todo projeto possa ser bem sucedido, devem se analisar, projetar e estudar minuciosamente e fazer o levantamento de todo os requisitos necessários para o projeto. Apartir das matérias de Modelagem e Desenvolvimento de Sistenas e Engenharia de Software, foi possível adquirir conhecimentos para um bom planejamento do trabalho. Como esse trabalho demonstrou ser extenso e complexo, decidimos por nos planejarmos a fim de evitar que possíveis erros de implementação fosse acarretado durante a fase de Desenvolvimento. Assim, fizemos uma reunião inicial para decidir os pontos importantes e decidir pontos sobre implementação do código. Utilizamos um organizador de tarefas no Excel para acompanhar as tarefas.Tal diagrama sera utilizado para as próximas etapas também. Veja abaixo, como ficou a organização do planejamento deste trabalho na figura abaixo.

Figura 1 – Planejamento TP Compiladores

(a) Diagrama Planejamento



Atividade	Inicio da Atividade	Duracao da Atividade	Inicio Atual	Duracao da Atividade	% de Concluido
Criar Repositorio no Git	1	1	1	1	100%
Reuniao Inicial	2	1	2	1	100%
Preparo da Modelagem	3	1	3	1	100%
Execucao do Trabalho	4	20	4	22	100%
Realizar a Documentacao	24	3	24	3	80%

Fonte: Desenvolvedores do Código

Tal arquivo se encontra dentro do diretório deste projeto criado pelo Programa Microsoft Excel, podendo ser acesso pelo caminho: ”TP_Compilador/Planejamento/

6. TESTES REALIZADOS

Para testar todo o compilador construído, foram propostos sete exemplos de teste, abordando algumas possibilidade de programas a serem compilados. Apartir desses arquivos podemos visualizar os seguintes retorno do compilador :

```

init
a, b, $c, is integer;
result is real;
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);
stop
```

Listing 6.1: Teste1.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```
**** Tokens lidos ****
< identifier, a >
< virgula >
< virgula >
< identifier, b_1 >
< virgula >
< identifier, b_2 >
< virgula >
< identifier, cc >
< virgula >
< num, 10 >
Error(1): Token ' ' inválido
Error(1): Token ':' inválido
< integer_type >
< ponto_virgula >
< init_program >
< write >
< abre_parent >
< literal, "Entre com o valor de a:" >
< fecha_parent >
< ponto_virgula >
< read >
```

```
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< identifier, b_1 >
< assign >
< identifier, a >
< mult_mulop >
< identifier, a >
< ponto_virgula >
< write >
< abre_parent >
< literal, "O valor de b1 e:" >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, b_1 >
< fecha_parent >
< ponto_virgula >
< identifier, b_2 >
< equal_relop >
< identifier, b >
< soma_addop >
< identifier, a >
< div_mulop >
< num, 2 >
< mult_mulop >
< abre_parent >
< identifier, a >
< soma_addop >
< num, 5 >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< literal, "O valor de b1 e:" >
< fecha_parent >
< ponto_virgula >
< identifier, Write >
< abre_parent >
< identifier, b1 >
< fecha_parent >
< ponto_virgula >
< stop_program >
```

```
70
71 **** Tabela de símbolos ****
72 Entrada          |                Mais info
73 string
74 Write
75 not
76 end
77 is
78 b1
79 or
80 b_2
81 b_1
82 real
83 if
84 cc
85 while
86 do
87 stop
88 read
89 init
90 else
91 integer
92 write
93 begin
94 b
95 a
96 and
```

Listing 6.2: Saida para o Codigo de teste : Teste1.txt

```
1 a, _valor, b_1, b_2, cc, 10_a : integer;
2 init
3 write("Entre com o valor de a:");
4 read (a);
5 b_1 := a * a;
6 write("O valor de b1 e:");
7 write (b_1);
8 b_2 = b + a/2 * (a + 5);
9 write("O valor de b1 e:");
10 Write (b1);
11 stop
```

Listing 6.3: Teste2.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```
1 -------Tokens Reconhecidos pelo Compilador-------
2 < identifier, a >
3 < virgula >
```

```
< virgula >
< identifier, b_1 >
< virgula >
< identifier, b_2 >
< virgula >
< identifier, cc >
< virgula >
< num, 10 >
Error(1): Token ' ' inválido
Error(1): Token ':' inválido
< integer_type >
< ponto_virgula >
< init_program >
< write >
< abre_parent >
< literal, "Entre com o valor de a:" >
< fecha_parent >
< ponto_virgula >
< read >
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< identifier, b_1 >
< assign >
< identifier, a >
< mult_mulop >
< identifier, a >
< ponto_virgula >
< write >
< abre_parent >
< literal, "O valor de b1 e:" >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, b_1 >
< fecha_parent >
< ponto_virgula >
< identifier, b_2 >
< equal_relop >
< identifier, b >
< soma_addop >
< identifier, a >
< div_mulop >
< num, 2 >
< mult_mulop >
```

```
< abre_parent >
< identifier, a >
< soma_addop >
< num, 5 >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< literal, "O valor de b1 e:" >
< fecha_parent >
< ponto_virgula >
< identifier, Write >
< abre_parent >
< identifier, b1 >
< fecha_parent >
< ponto_virgula >
< stop_program >



------------- Imprimindo a Tabela de símbolos-------------
                Entrada
string
Write
not
end
is
b1
or
b_2
b_1
real
if
cc
while
do
stop
read
init
else
integer
write
begin
b
a
and
```

Listing 6.4: Saida para o Codigo de teste : Teste1.txt

```
% Programa de Teste
Calculo de idade%
INIT
cont is int;
media, idade, soma is integer;
begin
cont := 5;
soma := 0;
do
write("Altura:" );
read (altura);
soma := soma+altura;
cont := cont - 1;
while(cont > 0)
media := soma / qts
write("Media: ");
write (media);
STOP
```

Listing 6.5: Teste3.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```

-------Tokens Reconhecidos pelo Compilador-------
< identifier, INIT >
< identifier, cont >
< is_decl >
< identifier, int >
< ponto_virgula >
< identifier, media >
< virgula >
< identifier, idade >
< virgula >
< identifier, soma >
< is_decl >
< integer_type >
< ponto_virgula >
< begin >
< identifier, cont >
< assign >
< num, 5 >
< ponto_virgula >
< identifier, soma >
< assign >
```

```
< num, 0 >
< ponto_virgula >
< do >
< write >
< abre_parent >
< literal, "Altura:" >
< fecha_parent >
< ponto_virgula >
< read >
< abre_parent >
< identifier, altura >
< fecha_parent >
< ponto_virgula >
< identifier, soma >
< assign >
< identifier, soma >
< soma_addop >
< identifier, altura >
< ponto_virgula >
< identifier, cont >
< assign >
< identifier, cont >
< menos_addop >
< num, 1 >
< ponto_virgula >
< while >
< abre_parent >
< identifier, cont >
< greater_than_relop >
< num, 0 >
< fecha_parent >
< identifier, media >
< assign >
< identifier, soma >
< div_mulop >
< identifier, qts >
< write >
< abre_parent >
< literal, "Media: " >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, media >
< fecha_parent >
< ponto_virgula >
< identifier, STOP >
```

```
70
71
72
73 ------------- Imprimindo a Tabela de símbolos-------------
74                  Entrada
75 STOP
76 string
77 int
78 INIT
79 cont
80 not
81 end
82 altura
83 idade
84 media
85 is
86 or
87 qts
88 soma
89 real
90 if
91 while
92 do
93 stop
94 read
95 init
96 else
97 integer
98 write
99 begin
100 and
```

Listing 6.6: Saida para o Codigo de teste : Teste3.txt

```
1 init
2 % Outro programa de teste
3 i, j, k, @total is integer;
4 nome is string
5 write("Digite o seu nome: ");
6 read(nome);
7 write("Digite um valor inteiro: );
8 read (I);
9 k := i * (5-i * 50 / 10;
10 j := i * 10;
11 k := i* j / k;
12 k := 4 + a $;
13 write(nome);
14 write(", os números gerados sao: ");
```

```
write(i);
write(j);
write(k);
```

Listing 6.7: Teste4.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```
-------Tokens Reconhecidos pelo Compilador-------
< init_program >
< mult_mulop >
< abre_parent >
< num, 5 >
< menos_addop >
< identifier, i >
< mult_mulop >
< num, 50 >
< div_mulop >
< num, 10 >
< ponto_virgula >
< identifier, j >
< assign >
< identifier, i >
< mult_mulop >
< num, 10 >
< ponto_virgula >
< identifier, k >
< assign >
< identifier, i >
< mult_mulop >
< identifier, j >
< div_mulop >
< identifier, k >
< ponto_virgula >
< identifier, k >
< assign >
< num, 4 >
< soma_addop >
< identifier, a >
Error(12): Token '$' inválido
< ponto_virgula >
< write >
< abre_parent >
< identifier, nome >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
```

```
< literal, ", os números gerados sao: " >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, i >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, j >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, k >
< fecha_parent >
< ponto_virgula >



------------- Imprimindo a Tabela de símbolos-------------
                Entrada
string
not
end
is
or
real
if
while
do
stop
k
nome
j
read
i
init
else
integer
write
begin
a
and
```

Listing 6.8: Saida para o Codigo de teste : Teste4.txt

```
init
a, b, c, maior is integer;
write("Digite uma idade: ");
read(a);
write("Digite outra idade: ");
read(b);
write("Digite mais uma idade: ");
read(c;
maior := 0;
if ( a>b and a>c )
maior := a;
else
if (b>c)
maior := b;
else
maior := c;
write("Maior idade: ");
write(maior);
end
```

Listing 6.9: Teste6.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```
-------Tokens Reconhecidos pelo Compilador-------
< init_program >
< identifier, a >
< virgula >
< identifier, b >
< virgula >
< identifier, c >
< virgula >
< identifier, maior >
< is_decl >
< integer_type >
< ponto_virgula >
< write >
< abre_parent >
< literal, "Digite uma idade: " >
< fecha_parent >
< ponto_virgula >
< read >
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
```

```
< literal, "Digite outra idade: " >
< fecha_parent >
< ponto_virgula >
< read >
< abre_parent >
< identifier, b >
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< literal, "Digite mais uma idade: " >
< fecha_parent >
< ponto_virgula >
< read >
< abre_parent >
< identifier, c >
< ponto_virgula >
< identifier, maior >
< assign >
< num, 0 >
< ponto_virgula >
< if >
< abre_parent >
< identifier, a >
< greater_than_relop >
< identifier, b >
< and_mulop >
< identifier, a >
< greater_than_relop >
< identifier, c >
< fecha_parent >
< identifier, maior >
< assign >
< identifier, a >
< ponto_virgula >
< else >
< if >
< abre_parent >
< identifier, b >
< greater_than_relop >
< identifier, c >
< fecha_parent >
< identifier, maior >
< assign >
< identifier, b >
< ponto_virgula >
< else >
```

```
72 < identifier, maior >
73 < assign >
74 < identifier, c >
75 < ponto_virgula >
76 < write >
77 < abre_parent >
78 < literal, "Maior idade: " >
79 < fecha_parent >
80 < ponto_virgula >
81 < write >
82 < abre_parent >
83 < identifier, maior >
84 < fecha_parent >
85 < ponto_virgula >
86 < end >
87
88
89
90 ------------- Imprimindo a Tabela de símbolos-------------
91                 Entrada
92 string
93 maior
94 not
95 end
96 is
97 or
98 real
99 if
100 while
101 do
102 stop
103 read
104 init
105 else
106 integer
107 write
108 c
109 begin
110 b
111 a
112 and
```

Listing 6.10: Saida para o Codigo de teste : Teste6.txt

```
1 % Programa de Teste%
2 Calculo de idade
3 init
4   cont_ is int;
```

```
  media, idade, soma_ is integer;
begin
  cont_ = 5;
  soma = 0;

  do
    write(Altura: );
    read (altura);
    soma := soma altura;
    cont_ := cont_ - 1;
  while(cont_ > 0)

  write(Media: );
  write (soma / qtd);

stop
```

Listing 6.11: Teste7.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```
-------Tokens Reconhecidos pelo Compilador-------
< identifier, Calculo >
< identifier, de >
< identifier, idade >
< init_program >
< identifier, cont_ >
< is_decl >
< identifier, int >
< ponto_virgula >
< identifier, media >
< virgula >
< identifier, idade >
< virgula >
< identifier, soma_ >
< is_decl >
< integer_type >
< ponto_virgula >
< begin >
< identifier, cont_ >
< equal_relop >
< num, 5 >
< ponto_virgula >
< identifier, soma >
< equal_relop >
< num, 0 >
< ponto_virgula >
< do >
```

```
< write >
< abre_parent >
Error(11): Token '' inválido
< identifier, Altura >
Error(11): Token ':' inválido
< fecha_parent >
< ponto_virgula >
< read >
< abre_parent >
< identifier, altura >
< fecha_parent >
< ponto_virgula >
< identifier, soma >
< assign >
< identifier, soma >
< identifier, altura >
< ponto_virgula >
< identifier, cont_ >
< assign >
< identifier, cont_ >
< menos_addop >
< num, 1 >
< ponto_virgula >
< while >
< abre_parent >
< identifier, cont_ >
< greater_than_relop >
< num, 0 >
< fecha_parent >
< write >
< abre_parent >
Error(17): Token '' inválido
< identifier, Media >
Error(17): Token ':' inválido
Error(17): Token '' inválido
< fecha_parent >
< ponto_virgula >
< write >
< abre_parent >
< identifier, soma >
< div_mulop >
< identifier, qtd >
< fecha_parent >
< ponto_virgula >
< stop_program >

```

```
75
76 ------------- Imprimindo a Tabela de símbolos-------------
77                  Entrada
78 string
79 int
80 not
81 end
82 altura
83 idade
84 media
85 is
86 or
87 Media
88 soma
89 real
90 if
91 while
92 Altura
93 do
94 stop
95 read
96 init
97 soma_
98 else
99 integer
100 write
101 qtd
102 de
103 begin
104 cont_
105 and
106 Calculo
```

Listing 6.12: Saida para o Codigo de teste : Teste7.txt

```
1 init
2   a, b is integer;
3   if( a >= b)
4   read(a);
5   end;
6 stop
```

Listing 6.13: Teste8.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```
1 -------Tokens Reconhecidos pelo Compilador-------
2 < init_program >
3 < identifier, a >
```

```
< virgula >
< identifier, b >
< is_decl >
< integer_type >
< ponto_virgula >
< if >
< abre_parent >
< identifier, a >
< greater_equals_relop >
< identifier, b >
< fecha_parent >
< read >
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< end >
< ponto_virgula >
< stop_program >



------------- Imprimindo a Tabela de símbolos-------------
                Entrada
string
not
end
is
or
real
if
while
do
stop
read
init
else
integer
write
begin
b
a
and
```

Listing 6.14: Saida para o Codigo de teste : Teste8.txt

```
init
  read(a);
```

```
  if(a == b) begin
    read(a);
  end;
stop
```

Listing 6.15: Teste9.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```


-------Tokens Reconhecidos pelo Compilador-------
< init_program >
< read >
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< if >
< abre_parent >
< identifier, a >
< equal_relop >
< equal_relop >
< identifier, b >
< fecha_parent >
< begin >
< read >
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< end >
< ponto_virgula >
< stop_program >



------------- Imprimindo a Tabela de símbolos-------------
                Entrada
string
not
end
is
or
real
if
while
do
stop
```

```
read
init
else
integer
write
begin
b
a
and
```

Listing 6.16: Saida para o Codigo de teste : Teste9.txt

```
init
  a;
  %Declarando errado  avariavel a%

  read(a);

stop
```

Listing 6.17: Teste10.txt

Para o codigo acima o compilador obeteve a seguinte saida :

```

-------Tokens Reconhecidos pelo Compilador-------
< init_program >
< identifier, a >
< ponto_virgula >
< read >
< abre_parent >
< identifier, a >
< fecha_parent >
< ponto_virgula >
< stop_program >




------------- Imprimindo a Tabela de símbolos-------------
                Entrada
string
not
end
is
or
real
if
while
do
```

```
stop
read
init
else
integer
write
begin
a
and
```

Listing 6.18: Saida para o Codigo de teste : Teste10.txt

7. METODOLOGIA

Como dito anteriormente, este trabalho está sendo desenvolvido em consonância com os conceitos ensinados pela orientadora e pelo Autor Aho, pelo livro : Compiladores, Técnicas e ferramentas.

```
/*
Na etapa 1, o compilador deverá exibir a sequência de tokens identificados e os símbolos (identificadores e
palavras reservadas) instalados na Tabela de Símbolos. Nas etapas seguintes, isso não deverá ser exibido.
 */
//Commit para reparar possíveis bugs//

import analisador_lexico.Lexer;
import analisador_lexico.Tag;
import analisador_lexico.Token;
import exception.InvalidTokenException;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
import java.util.ArrayList;

public class Main {


  public static void main(String[] args) {
    ArrayList<Token> tokens = new ArrayList<Token> ();
    Lexer L = null;
    int line = -5;
    try {
      L = new Lexer("codigos_teste/Teste10.txt");

      L.adicionapalavras();//Inicia adicionando palavras reservadas
      System.out.println("-------Tokens Reconhecidos pelo Compilador -------");
      Token T = new Token(0, line);
      while (T.tag != Tag.EOF) {
        try {
          T = L.scan();
          if(T.tag == Tag.EOF)
            break;
          T.imprimeToken(T);
          tokens.add(T);
          line = T.line;
```

```
38          } catch (InvalidTokenException | IOException e) {
39            System.out.println(e.getMessage());
40            try {
41              L.readch();
42            } catch (IOException e1) {
43              e1.printStackTrace();
44            }
45          }
46        }
47        line++;
48        tokens.add(new Token(Tag.EOF, line));
49        L.imprimirTabela();
50      } catch (FileNotFoundException e) {
51        e.printStackTrace();
52      }
53    }
54  }
```

Codigos/Main.java

8. SUGESTÕES PARA TRABALHOS FUTUROS

Nesta etapa, finalizamos o processo de analise Léxica do Compilador (Etapa 1), ficando restando 2 etapas, aos quais deverão ser implementados na parte 2 o analisador Sintático e na parte 3 o analisador Semântico e o Gerador de código. Para as próximas etapas, fica sugerido que devemos implementar a recuperação de Erros para que seja possível implementar um compilador mais eficiente e próximo da realidade.

9. REFERÊNCIAS

9.1. LIVROS

A. V. Aho, R. Sethi, J. D. Ullman Compiladores: Princípios, técnicas e ferramentas LTC - Livros Técnicos e Científicos Editora, 2013

9.2. PROGRAMAS

- Apache Netbeans
- Visual Studio
- Microsoft Office Excel

9.3. SITE

- Site : Overleaf Para criação do relatório