

Analizador Léxico - 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");
```

- 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

A abordagem utilizada foi uma abordagem semelhante a apresentada pelo livro texto da disciplina (Compilers - Principles Techniques and Tools by Alfred Aho - Monica Lam- Ravi Sethi- Jeffrey Ullman - Second Edition).

- Principais Classes do Compilador**
 - Class Tag**
 - A classe Tag possui uma estrutura do tipo **enum** enumerando todos os possíveis tipos de Tag que podem ser atribuídos a Tokens.
 - 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

- **Class Token**

- A classe Token possui os seguintes atributos: **tag**, responsável por salvar o valor de tag daquele Token, e **reserved**, responsável por dizer se o Token é uma palavra reservada ou não para fins de facilitar na hora de organizar os prints. Além disso, possui um método construtor e um método **toString()**, que apenas retorna o valor da tag em formato de string.

- **Class Word**

- A classe Word é uma classe que herda da classe Token, e serve para representar Tokens que possuem o lexema formado por letras ou operadores. Além de herdar os atributos da classe Token, a classe Word possui ainda o atributo **lexeme**, que serve para armazenar os respectivos lexemas de cada Token. Além disso, na classe Word também são criados os objetos estáticos do tipo Word que representam as palavras reservadas e operadores da linguagem, tais objetos estáticos são instanciados em “Word.h” e tem seus valores definidos em “Word.cpp”.

- **Class Num**

- A classe Num é uma classe que herda da classe Token, e serve para representar Tokens numéricos do tipo inteiro. Além de herdar os atributos da classe Token, a classe Num possui ainda o atributo **value**, do tipo int, que serve para armazenar os respectivos valores de cada Token. Além disso, todos os Tokens do tipo Num possuem uma tag de valor “NUM”, sendo assim o construtor da classe já possui esse parâmetro definido, assim como é implementado no livro.

- **Class Real**

- A classe Real é uma classe que herda da classe Token, e serve para representar Tokens numéricos de tipo flutuante. Além de herdar os atributos da classe Token, a classe Real possui ainda o atributo **value**, do tipo float, que serve para armazenar os respectivos valores de cada Token. Além disso, todos os Tokens do tipo Real possuem uma tag de valor “REAL”, sendo assim o construtor da classe já possui esse parâmetro definido, assim como é implementado no livro.

- **Class Lexer**

- A classe Lexer é a classe que representa o Analisador Léxico, portanto, a classe principal dessa etapa do trabalho. Na classe Lexer existem os seguintes atributos:
 - **file** → Leitor de Arquivo
 - **line** → Contador de linhas do arquivo fonte
 - **ch** → Variável que recebe o caractere lido do arquivo fonte
 - **TabelaDeSímbolos** → Tabela de Símbolos
 - A Tabela de Símbolos instanciada como atributo da classe Lexer será a Tabela de Símbolos geral do compilador. Futuramente, na implementação do Analisador Sintático usaremos uma referência para a Tabela de Símbolos da classe Lexer.
- Além disso, a classe também possui os seguintes métodos:
 - **addSymbol** → Adiciona Tokens na Tabela de Símbolos
 - **readch** → Lê o próximo caractere do arquivo fonte
 - **readch(char c)** → Lê o próximo caractere do arquivo fonte e checka se é igual ao caractere passado como parâmetro
 - **scan()** → Faz a análise dos caracteres e forma os Tokens

- **Class Main**

- A classe Main é a classe que usaremos para instanciar o Analisador Léxico e chamar o método scan() para realizar a análise do arquivo e criação dos Tokens. Nessa classe temos um loop que chama o método scan() e recebe os objetos Token retornados até o Analisador Léxico atingir o fim do arquivo fonte, e dentro desse loop checamos se ocorreu algum erro ao abrir o arquivo ou se ocorreu algum erro léxico durante a análise do arquivo fonte. Além disso, essa classe possui mais dois loops, um para percorrer a Tabela de Símbolos printar as palavras reservadas salvas na tabela, e o outro para percorrer a Tabela de Símbolos e printar os identificadores salvos na tabela. E por fim, um print indicando que a análise léxica foi bem sucedida.

- **Principais Métodos do Compilador**

- Lexer.cpp → Token scan()
 - O método scan() da classe Lexer é responsável por fazer a análise do(s) caractere(s) lido(s) do artigo fonte. A partir da análise feita a classe scan() cria e retorna um objeto Token com o(s) caractere(s) lido(s). Caso o Token formado seja um Token do tipo “identifier” ele também será adicionado na tabela de símbolos dentro do flow do método scan(). Além disso, antes de retornar qualquer Token identificado, o método scan() possui prints antes dos comandos de **return**, para que fique mais visível na saída do programa a ordem em que os tokens foram identificados.
 - O método scan() reconhece todos os tokens que possuem tags específicas para eles, como mostrado na tabela de tags acima, para os demais caracteres ele cria objetos do tipo Token com tags que referem o número daquele caractere na tabel ASCII.
- Lexer.cpp → addSymbol()

- O método `addSymbol()` é responsável por adicionar um Token do tipo “identifier” na Tabela de Símbolos caso ele já não esteja presente na Tabela.
- `Lexer.cpp` → `readch()`
 - O método `readch()` é responsável por ler o próximo caractere do arquivo fonte e salvar esse caractere no atributo “ch” da classe `Lexer`.
 - `Lexer.cpp` → `readch(char c)`
 - O método `readch(char c)` chama o método `readch()`, e além disso realiza uma checagem para ver se a variável salva em “ch” por `readch()` é igual ao caractere passado por parâmetro para `readch(char c)`, caso a igualdade seja garantida a variável “ch” é limpa e o método retorna **true**, caso contrário é retornado **false**.

3 – Resultado dos testes realizados

- **Teste1.txt**

Programa Fonte (Antes):

```
programa teste1

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

Saída Esperada (Antes): Nesse teste, a saída esperada é um erro léxico apontado na linha 8, onde ocorre a má formação de um Token do tipo **float_const**, pois após o ponto (“.”) deveria ter pelo menos mais um dígito numérico.

Saída do Compilador (Antes):

```
● 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
LEXEME: programa TAG: 282
LEXEME: teste1 TAG: 282
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: b TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: result TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: x TAG: 282
LEXEME: is TAG: 267
LEXEME: float TAG: 269
LEXEME: ; TAG: 59
LEXEME: begin TAG: 265
LEXEME: a TAG: 282
LEXEME: = TAG: 61
LEXEME: 12 TAG: 280
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: x TAG: 282
LEXEME: = TAG: 61
TOKEN BADLY BUILT IN LINE 8
EXPECTED A NUMERICAL CHARACTER AFTER '.' - FOUND: ;
ENDING RUN DUE TO LEXICAL ERROR !!!
○ PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (Depois):

```
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 (Depois): Para consertar o erro mostrado acima, alterei a linha 8 do arquivo fonte para “ x = 12.1; “, ou seja, agora o token do tipo **float_const** está sendo formado corretamente. Sendo assim, agora esperasse que o arquivo todo seja analisado e todos os tokens sejam mostrados na saída.

Saída do Compilador (Depois):

```
• 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
LEXEME: programa TAG: 282
LEXEME: teste1 TAG: 282
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: b TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: result TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: x TAG: 282
LEXEME: is TAG: 267
LEXEME: float TAG: 269
LEXEME: ; TAG: 59
LEXEME: begin TAG: 265
LEXEME: a TAG: 282
LEXEME: = TAG: 61
LEXEME: 12 TAG: 280
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: x TAG: 282
LEXEME: = TAG: 61
LEXEME: 12.1 TAG: 281
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: result TAG: 282
LEXEME: = TAG: 61
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: * TAG: 42
LEXEME: b TAG: 282
LEXEME: + TAG: 43
LEXEME: 1 TAG: 280
LEXEME: ) TAG: 41
LEXEME: / TAG: 47
LEXEME: ( TAG: 40
LEXEME: c TAG: 282
LEXEME: + TAG: 43
LEXEME: 2 TAG: 280
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: {Resultado: } TAG: 285
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: result TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: end TAG: 266
LEXEME: . TAG: 46
```

```
RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: a TAG: 282
LEXEME: b TAG: 282
LEXEME: c TAG: 282
LEXEME: programa TAG: 282
LEXEME: result TAG: 282
LEXEME: teste1 TAG: 282
LEXEME: x TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> |
```

- Teste2.txt

Programa Fonte:

```
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: Nesse teste não tem nenhum erro que o analisador léxico pode identificar. Logo, esperasse que o arquivo fonte seja analisado por completo.

Saída do Compilador:

```
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
LEXEME: program TAG: 264
LEXEME: teste2 TAG: 282
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: b TAG: 282
LEXEME: , TAG: 44
LEXEME: c TAG: 282
LEXEME: : TAG: 58
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: d TAG: 282
LEXEME: , TAG: 44
LEXEME: _ TAG: 95
LEXEME: var TAG: 282
LEXEME: : TAG: 58
LEXEME: float TAG: 269
LEXEME: ; TAG: 59
LEXEME: teste2 TAG: 282
LEXEME: = TAG: 61
LEXEME: 1 TAG: 280
LEXEME: ; TAG: 59
LEXEME: Read TAG: 282
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: b TAG: 282
LEXEME: = TAG: 61
LEXEME: a TAG: 282
LEXEME: * TAG: 42
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: c TAG: 282
LEXEME: = TAG: 61
LEXEME: b TAG: 282
LEXEME: + TAG: 43
LEXEME: a TAG: 282
LEXEME: 35 TAG: 280
LEXEME: / TAG: 47
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: c TAG: 282
LEXEME: ; TAG: 59
LEXEME: val TAG: 282
LEXEME: : TAG: 58
LEXEME: = TAG: 61
LEXEME: 34.2 TAG: 281
LEXEME: c TAG: 282
LEXEME: = TAG: 61
LEXEME: val TAG: 282
LEXEME: + TAG: 43
LEXEME: 2.2 TAG: 281
LEXEME: + TAG: 43
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: val TAG: 282
LEXEME: ) TAG: 41
LEXEME: end TAG: 266
LEXEME: . TAG: 46
```

```
RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: Read TAG: 282
LEXEME: a TAG: 282
LEXEME: b TAG: 282
LEXEME: c TAG: 282
LEXEME: d TAG: 282
LEXEME: teste2 TAG: 282
LEXEME: val TAG: 282
LEXEME: var TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- **Teste3.txt**

Programa Fonte (Antes):

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

Saída Esperada (Antes): No caso desse teste o Analisador Léxico não vai identificar nenhum erro e vai conseguir analisar o arquivo fonte por completo. As palavras reservadas “int” mal formadas nas linhas 6 e 7 serão reconhecidas como identificadores pelo Analisador Léxico, assim como as palavras que compõe o comentário mal formado na linha 8. Estes erros, junto com o erro da linha 13 serão tratados em etapas futuras.

Saída do Compilador (Antes):


```
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
LEXEME: program TAG: 264
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: aux TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: b TAG: 282
LEXEME: is TAG: 267
LEXEME: float TAG: 269
LEXEME: begin TAG: 265
LEXEME: b TAG: 282
LEXEME: = TAG: 61
LEXEME: 0 TAG: 280
LEXEME: ; TAG: 59
LEXEME: in TAG: 282
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: in TAG: 282
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: > TAG: 257
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: then TAG: 272
LEXEME: / TAG: 47
LEXEME: / TAG: 47
LEXEME: troca TAG: 282
LEXEME: variaveis TAG: 282
LEXEME: aux TAG: 282
LEXEME: = TAG: 61
LEXEME: b TAG: 282
● LEXEME: ; TAG: 59
LEXEME: b TAG: 282
LEXEME: = TAG: 61
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: a TAG: 282
LEXEME: = TAG: 61
LEXEME: aux TAG: 282
LEXEME: end TAG: 266
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
```

```
RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: a TAG: 282
LEXEME: aux TAG: 282
LEXEME: b TAG: 282
LEXEME: in TAG: 282
LEXEME: troca TAG: 282
LEXEME: variaveis TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
○ PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (Depois):

```
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 (Depois): Ao consertarmos as palavras reservadas “int” mal formadas nas linhas 6 e 7, e também arrumar o comentário de uma linha na linha 8, as palavras reservadas “int” serão consideradas palavras reservadas e vão constituir tokens com a tag da palavra reservada “int”, e o comentário será desconsiderado. Sendo assim, somente 3 identificadores serão identificados no arquivo fonte e salvos na Tabela de Símbolos.

Saída do Compilador (Depois):

```
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
LEXEME: program TAG: 264
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: aux TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: b TAG: 282
LEXEME: is TAG: 267
LEXEME: float TAG: 269
LEXEME: begin TAG: 265
LEXEME: b TAG: 282
LEXEME: = TAG: 61
LEXEME: 0 TAG: 280
LEXEME: ; TAG: 59
LEXEME: int TAG: 268
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: int TAG: 268
LEXEME: ( TAG: 40
● LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: > TAG: 257
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: then TAG: 272
LEXEME: aux TAG: 282
LEXEME: = TAG: 61
LEXEME: b TAG: 282
LEXEME: ; TAG: 59
LEXEME: b TAG: 282
LEXEME: = TAG: 61
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: a TAG: 282
LEXEME: = TAG: 61
LEXEME: aux TAG: 282
LEXEME: end TAG: 266
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
```

```
RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: a TAG: 282
LEXEME: aux TAG: 282
LEXEME: b TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
○ PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- **Teste4.txt**

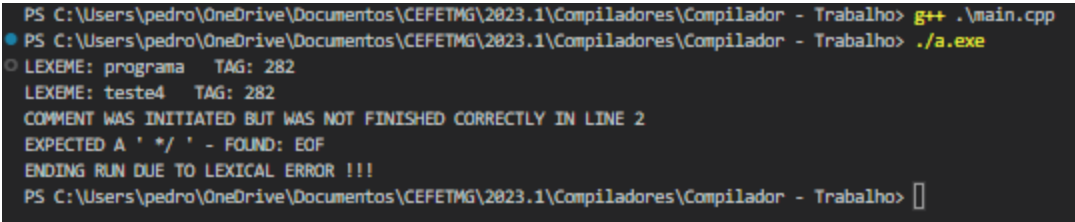
Programa Fonte (Antes):

```
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 (Antes): No caso desse teste, o Analisador Léxico vai identificar dois identificadores na primeira linha. Além disso, ele também vai identificar um comentário aberto na linha 2 que não é fechado. Portanto, será printado um erro léxico em relação a construção errada do comentário.

Saída do Compilador (Antes):



Programa Fonte (Depois):

```
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 (Depois): Quando consertamos o comentário e fechamos ele na linha 24, o Analisador Léxico reconhece os dois identificadores da primeira linha e ignora o comentário iniciado na segunda linha. Assim, o Analisador Léxico vai conseguir analisar o arquivo fonte sem erros.

Saída do Compilador (Depois):

```
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
LEXEME: programa TAG: 282
LEXEME: teste4 TAG: 282

RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: programa TAG: 282
LEXEME: teste4 TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> 
```

- Teste5.txt

Programa Fonte (Antes):

```
/* 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 (Antes): No caso desse teste é esperado que o Analisador Léxico identifique um erro na linha 28 na formação de um token do tipo **char_const**. O **char_const** na linha 28 deveria ser ‘n’, porém ele é formado sem a segunda aspas simples, acarretando em um erro léxico.

Saída do Compilador (Antes):

```
● 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
● LEXEME: program TAG: 264
LEXEME: teste5 TAG: 282
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: b TAG: 282
LEXEME: , TAG: 44
LEXEME: c TAG: 282
LEXEME: , TAG: 44
LEXEME: maior TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
● LEXEME: outro TAG: 282
LEXEME: is TAG: 267
LEXEME: char TAG: 270
LEXEME: ; TAG: 59
LEXEME: begin TAG: 265
LEXEME: repeat TAG: 274
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {A} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {B} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {C} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: > TAG: 257
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: && TAG: 263
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: > TAG: 257
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: ) TAG: 41
LEXEME: end TAG: 266
LEXEME: maior TAG: 282
LEXEME: = TAG: 61
LEXEME: a TAG: 282
```

```
LEXEME: else TAG: 273
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: > TAG: 257
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: then TAG: 272
LEXEME: maior TAG: 282
LEXEME: = TAG: 61
LEXEME: b TAG: 282
LEXEME: ; TAG: 59
LEXEME: else TAG: 273
LEXEME: maior TAG: 282
LEXEME: = TAG: 61
LEXEME: c TAG: 282
LEXEME: end TAG: 266
LEXEME: end TAG: 266
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {Maior valor:} TAG: 285
LEXEME: } TAG: 125
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: maior TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {Outro? (S/N)} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: outro TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: until TAG: 275
LEXEME: ( TAG: 40
LEXEME: outro TAG: 282
LEXEME: == TAG: 256
LEXEME: 'N' TAG: 284
LEXEME: || TAG: 262
LEXEME: outro TAG: 282
LEXEME: == TAG: 256
TOKEN BADLY BUILT IN LINE 28
EXPECTED ' AFTER THE ASCII CHARACTER - FOUND: )
ENDING RUN DUE TO LEXICAL ERROR !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

Programa Fonte (Depois):

```
/* 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 (Depois): Consertando o erro na linha 28 adicionando uma aspas simples após o caractere ‘n’, o Analisador Léxico será capaz de identificar todos os tokens do arquivo fonte sem reportar nenhum erro.

Saída do Compilador (Depois):

```
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
LEXEME: program TAG: 264
LEXEME: teste5 TAG: 282
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: b TAG: 282
LEXEME: , TAG: 44
LEXEME: c TAG: 282
LEXEME: , TAG: 44
LEXEME: maior TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: outro TAG: 282
LEXEME: is TAG: 267
LEXEME: char TAG: 270
LEXEME: ; TAG: 59
LEXEME: begin TAG: 265
LEXEME: repeat TAG: 274
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {A} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {B} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {C} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: > TAG: 257
LEXEME: b TAG: 282
LEXEME: ) TAG: 41
LEXEME: && TAG: 263
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: > TAG: 257
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: ) TAG: 41
LEXEME: end TAG: 266
LEXEME: maior TAG: 282
LEXEME: = TAG: 61
LEXEME: a TAG: 282
```

```
LEXEME: else TAG: 273
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: b TAG: 282
LEXEME: > TAG: 257
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: then TAG: 272
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {Maior valor:} TAG: 285
LEXEME: } TAG: 125
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: maior TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: {Outro? (S/N)} TAG: 285
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: outro TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: until TAG: 275
LEXEME: ( TAG: 40
LEXEME: outro TAG: 282
LEXEME: == TAG: 256
LEXEME: 'N' TAG: 284
LEXEME: || TAG: 262
LEXEME: outro TAG: 282
LEXEME: == TAG: 256
LEXEME: 'n' TAG: 284
LEXEME: ) TAG: 41
LEXEME: end TAG: 266
```

```
RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: a TAG: 282
LEXEME: b TAG: 282
LEXEME: c TAG: 282
LEXEME: maior TAG: 282
LEXEME: outro TAG: 282
LEXEME: teste5 TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- Teste6.txt

Programa Fonte:

```
/* Teste */
program teste6

    x, y is int;

    duda is char;

/* comentário 1*/

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

    if (x) then
        maior = a;
    end

    else then
        maior = b;
    end

/* comentário 2*/

end
```

Saída Esperada: No caso desse teste, o Analisador Léxico não irá identificar nenhum erro léxico e reconhecerá todos os tokens.

Saída do Compilador:


```
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
LEXEME: program TAG: 264
LEXEME: teste6 TAG: 282
LEXEME: x TAG: 282
LEXEME: , TAG: 44
LEXEME: y TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: duda TAG: 282
LEXEME: is TAG: 267
LEXEME: char TAG: 270
LEXEME: ; TAG: 59
LEXEME: begin TAG: 265
LEXEME: repeat TAG: 274
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: x TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: x TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: if TAG: 271
LEXEME: ( TAG: 40
LEXEME: x TAG: 282
LEXEME: ) TAG: 41
LEXEME: then TAG: 272
LEXEME: maior TAG: 282
LEXEME: = TAG: 61
LEXEME: a TAG: 282
LEXEME: ; TAG: 59
LEXEME: end TAG: 266
LEXEME: end TAG: 266

RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: a TAG: 282
LEXEME: duda TAG: 282
LEXEME: maior TAG: 282
LEXEME: teste6 TAG: 282
LEXEME: x TAG: 282
LEXEME: y TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
○ PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> █
```

- Teste7.txt

Programa Fonte:

```
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 );
    read(b);

    c = a + b + 5.2;

    write {c: };
    write (c);

end
end
end
```

Saída Esperada: No caso desse teste, o Analisador Léxico não irá identificar nenhum erro léxico e reconhecerá todos os tokens.

Saída do Compilador:

```
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho> ./a.exe
LEXEME: program TAG: 264
LEXEME: teste7 TAG: 282
LEXEME: program TAG: 264
LEXEME: teste TAG: 282
LEXEME: a TAG: 282
LEXEME: , TAG: 44
LEXEME: b TAG: 282
LEXEME: is TAG: 267
LEXEME: int TAG: 268
LEXEME: ; TAG: 59
LEXEME: c TAG: 282
LEXEME: is TAG: 267
LEXEME: float TAG: 269
LEXEME: ; TAG: 59
LEXEME: begin TAG: 265
LEXEME: a TAG: 282
LEXEME: = TAG: 61
LEXEME: 1 TAG: 280
LEXEME: ; TAG: 59
LEXEME: c TAG: 282
LEXEME: = TAG: 61
LEXEME: 1 TAG: 281
LEXEME: ; TAG: 59
LEXEME: read TAG: 278
LEXEME: ( TAG: 40
LEXEME: a TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: c TAG: 282
LEXEME: = TAG: 61
LEXEME: b TAG: 282
LEXEME: + TAG: 43
LEXEME: 5.2 TAG: 281
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: {c: } TAG: 285
LEXEME: ; TAG: 59
LEXEME: write TAG: 279
LEXEME: ( TAG: 40
LEXEME: c TAG: 282
LEXEME: ) TAG: 41
LEXEME: ; TAG: 59
LEXEME: end TAG: 266

RESERVED WORDS IN THE SYMBOL TABLE:
LEXEME: begin TAG: 265
LEXEME: char TAG: 270
LEXEME: do TAG: 277
LEXEME: else TAG: 273
LEXEME: end TAG: 266
LEXEME: float TAG: 269
LEXEME: if TAG: 271
LEXEME: int TAG: 268
LEXEME: is TAG: 267
LEXEME: program TAG: 264
LEXEME: read TAG: 278
LEXEME: repeat TAG: 274
LEXEME: then TAG: 272
LEXEME: until TAG: 275
LEXEME: while TAG: 276
LEXEME: write TAG: 279

IDENTIFIERS IN THE SYMBOL TABLE:
LEXEME: a TAG: 282
LEXEME: b TAG: 282
LEXEME: c TAG: 282
LEXEME: teste TAG: 282
LEXEME: teste7 TAG: 282

LEXICAL ANALYSIS COMPLETE !!!
PS C:\Users\pedro\OneDrive\Documentos\CEFETMG\2023.1\Compiladores\Compilador - Trabalho>
```