

# Trabalho Compiladores

## Analizador Léxico - pasC

Professor: Gustavo Fernandes

Integrantes:

Marcos Magno de Carvalho  
Marcos Vinicius Gonçalves Cunha

## Package compiladores

---

- **compiladores.lexer**

- **Classes**

- **Token**

- Esta classe implementa o analisador léxico.

- **Métodos**

- **init**

- Método construtor. Iniciar às variáveis e a abertura do arquivo.

- **file\_pointer()**

- Esse método manipula o ponteiro do arquivo em leitura com os métodos da biblioteca File:

- seek(inteiro) : Começa a ler o arquivo a partir da posição passada por parâmetro.

- tell(): Retorna um inteiro que representa a posição atual do ponteiro;

- Obs.: O método é utilizado para voltar uma posição do ponteiro na leitura do arquivo em alguns momentos.

- **panic\_mode(linha, coluna, tipo de erro)**

- Esse método é responsável por criar um vetor com os erros ocorrido durante a análise léxica

- .

- Parâmetros:

- linha: inteiro - Número da linha que ocorreu um erro.

- coluna: inteiro - Número da coluna que ocorreu um erro.

- tipo de erro: string - Texto com o erro que será apresentado para o usuário.

- **get\_erro()**

- Esse método apenas percorrer o vetor de erros e imprime na tela do usuários os erros.

- **close\_file()**

- Responsável por fechar o arquivo quando encontrar o fim do mesmo ou quando o programa retornar None.

- **next\_token(nome do arquivo, tabela de símbolo)**

- Esse método é responsável por fazer a leitura do arquivo ( lê-se um caracter a cada iteração) até que se encontre o fim do arquivo ou retorne um None.

- Dentro desse método é verificado cada estado do autômato e caso se encontre um token válido o mesmo retorna um objeto do tipo Token.

Caso contrário é chamado o `panic_mode()` para sinalizar a mensagem de erro e continua-se a leitura do arquivo para o/os próximo/próximos arquivos, até que se encontre o lexema desejado para formar o token. Caso o token não seja formado até o fim do arquivo, retorna EOF.

Parâmetros:

nome do arquivo: string - Nome do arquivo que será analisado.

tabela de símbolo: objeto - Objeto que representa a tabela de símbolo.

- **main()**

Método principal, responsável por criar um objeto do tipo Lexer bem como Tabela de Símbolo.

Esse método é composto também por um while que verifica a variável *var* até que ela se torne False ( quando se chega no fim do arquivo ou obter None). Enquanto se for verdade, a variável token solicita um novo token e o imprime na tela.

Aqui também se obtém a tabela de símbolo e às mensagens de erros.

- **compiladores.symbol\_table**

- **Classe**

- **TabelaSimbolo**

Essa classe implementa a tabela de símbolos e a manipula;

- **Métodos**

- **init**

Método construtor. Iniciar a tabela de símbolo e insere as palavras chaves ( KW ).

- **put\_symbol\_table(w)**

Método responsável por inserir na tabela de símbolo.

Parâmetros:

w: objeto - Objeto do tipo Token

- **set\_identifier()**

Responsável por incrementar a variável identifier, utilizada para servir de Keys na hash tabela de símbolo.

- **get\_identifier()**

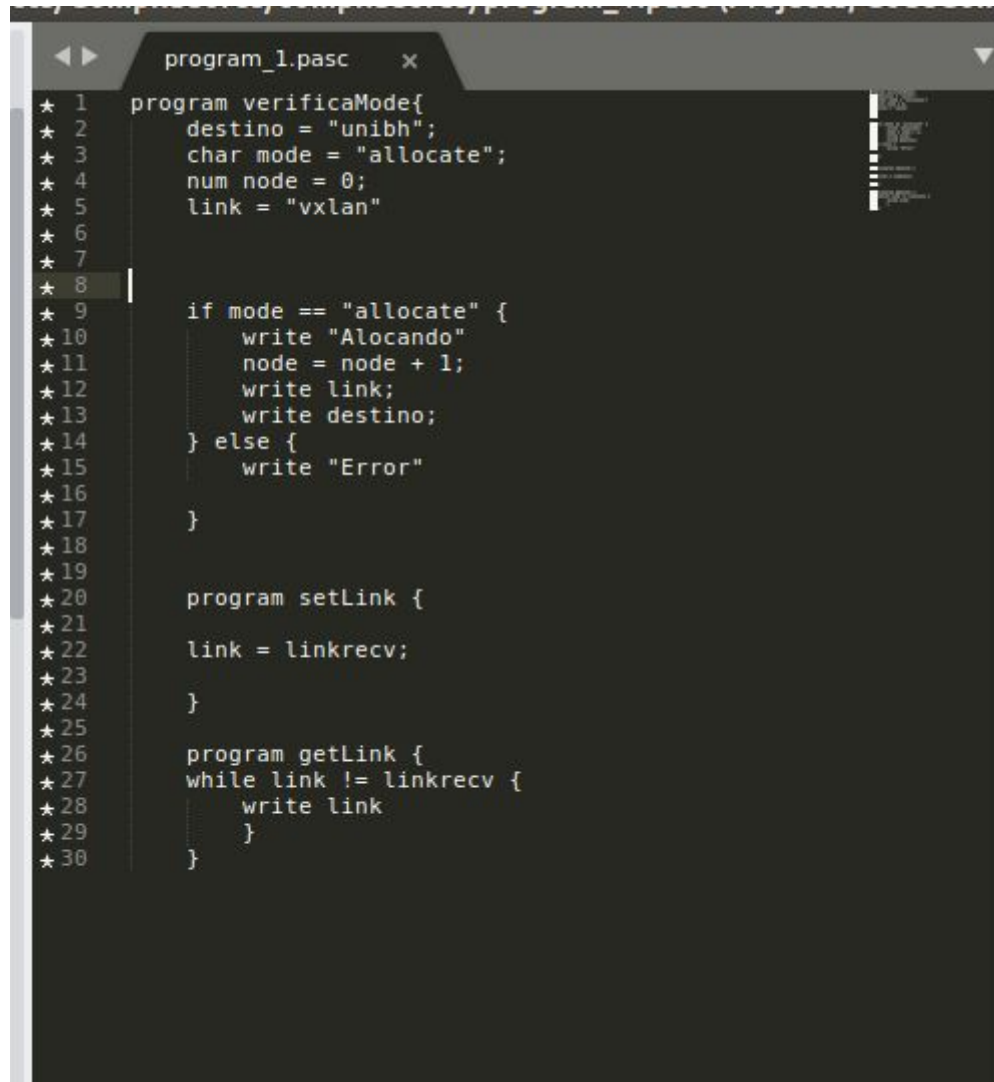
Retorna a variável identifier.

- **get\_token()**  
Percorre a tabela de símbolo e verificar se já existe o lexema cadastrado. Caso haja retorna o objeto token, caso contrário, retorna None.
  - **get\_ts()**  
Percorrer a tabela de símbolo e retorna tokens cadastrados.
- **compiladores.tag**
  - **Classe**
    - **Tag\_type()**  
Responsável por criar às variáveis de operadores, símbolos, palavra chave, ID, Literal, constante numérica e constante literal.
- **compiladores.token**
  - **Classe**
    - **Token()**  
Essa classe implementa os atributos dos token
  - **Métodos**
    - **init**  
Método construtor. Inicia classe, lexema, linha e coluna.
    - **getLexema()**  
Retorna o lexema
    - **getClass()**  
Retorna a classe
    - **toString()**  
Converte o objeto em um padrão de string, contendo a classe e o lexema.

## Report

---

### Programa Correto 1:



```
program_1.pasc x
★ 1  program verificaMode{
★ 2      destino = "unibh";
★ 3      char mode = "allocate";
★ 4      num node = 0;
★ 5      link = "vxlan"
★ 6
★ 7
★ 8
★ 9      if mode == "allocate" {
★10          write "Alocando"
★11          node = node + 1;
★12          write link;
★13          write destino;
★14      } else {
★15          write "Error"
★16
★17      }
★18
★19
★20      program setLink {
★21
★22          link = linkrecv;
★23
★24      }
★25
★26      program getLink {
★27          while link != linkrecv {
★28              write link
★29          }
★30      }
```

Tokens:

```
root@vm-pc: /opt/Projects/Compiladores/compiladores 102x65
Token: < ID, "verificaMode" Linha: 1 Coluna: 21
Token: < SMB_OBC, "{" Linha: 1 Coluna: 22
Token: < ID, "destino" Linha: 2 Coluna: 12
Token: < OP_ASS, "=" Linha: 2 Coluna: 14
Token: < LIT, "unibh" Linha: 2 Coluna: 22
Token: < SMB_SEM, ";" Linha: 2 Coluna: 23
Token: < KW, "char" Linha: 3 Coluna: 9
Token: < ID, "mode" Linha: 3 Coluna: 14
Token: < OP_ASS, "=" Linha: 3 Coluna: 16
Token: < LIT, "allocate" Linha: 3 Coluna: 27
Token: < SMB_SEM, ";" Linha: 3 Coluna: 28
Token: < KW, "num" Linha: 4 Coluna: 8
Token: < ID, "node" Linha: 4 Coluna: 13
Token: < OP_ASS, "=" Linha: 4 Coluna: 15
Token: < CON_NUM, "0" Linha: 4 Coluna: 17
Token: < SMB_SEM, ";" Linha: 4 Coluna: 18
Token: < ID, "link" Linha: 5 Coluna: 9
Token: < OP_ASS, "=" Linha: 5 Coluna: 11
Token: < LIT, "vxlan" Linha: 5 Coluna: 19
Token: < KW, "if" Linha: 9 Coluna: 7
Token: < ID, "mode" Linha: 9 Coluna: 12
Token: < OP_EQ, "==" Linha: 9 Coluna: 15
Token: < LIT, "allocate" Linha: 9 Coluna: 26
Token: < SMB_OBC, "{" Linha: 9 Coluna: 28
Token: < KW, "write" Linha: 10 Coluna: 14
Token: < LIT, "Alocando" Linha: 10 Coluna: 25
Token: < ID, "node" Linha: 11 Coluna: 13
Token: < OP_ASS, "=" Linha: 11 Coluna: 15
Token: < ID, "node" Linha: 11 Coluna: 20
Token: < OP_AD, "+" Linha: 11 Coluna: 22
Token: < CON_NUM, "1" Linha: 11 Coluna: 24
Token: < SMB_SEM, ";" Linha: 11 Coluna: 25
Token: < KW, "write" Linha: 12 Coluna: 14
Token: < ID, "link" Linha: 12 Coluna: 19
Token: < SMB_SEM, ";" Linha: 12 Coluna: 20
Token: < KW, "write" Linha: 13 Coluna: 14
Token: < ID, "destino" Linha: 13 Coluna: 22
Token: < SMB_SEM, ";" Linha: 13 Coluna: 23
Token: < SMB_CBC, "}" Linha: 14 Coluna: 6
Token: < KW, "else" Linha: 14 Coluna: 11
Token: < SMB_OBC, "{" Linha: 14 Coluna: 13
Token: < KW, "write" Linha: 15 Coluna: 14
Token: < LIT, "Error" Linha: 15 Coluna: 22
Token: < SMB_CBC, "}" Linha: 17 Coluna: 6
Token: < KW, "program" Linha: 20 Coluna: 12
Token: < ID, "setLink" Linha: 20 Coluna: 20
Token: < SMB_OBC, "{" Linha: 20 Coluna: 22
Token: < ID, "link" Linha: 22 Coluna: 9
Token: < OP_ASS, "=" Linha: 22 Coluna: 11
Token: < ID, "linkrecv" Linha: 22 Coluna: 20
Token: < SMB_SEM, ";" Linha: 22 Coluna: 21
Token: < SMB_CBC, "}" Linha: 24 Coluna: 6
Token: < KW, "program" Linha: 26 Coluna: 12
Token: < ID, "getLink" Linha: 26 Coluna: 20
Token: < SMB_OBC, "{" Linha: 26 Coluna: 22
Token: < KW, "while" Linha: 27 Coluna: 10
Token: < ID, "link" Linha: 27 Coluna: 15
Token: < OP_NE, "!=" Linha: 27 Coluna: 18
Token: < ID, "linkrecv" Linha: 27 Coluna: 27
Token: < SMB_OBC, "{" Linha: 27 Coluna: 29
Token: < KW, "write" Linha: 28 Coluna: 14
Token: < ID, "link" Linha: 28 Coluna: 19
Token: < SMB_CBC, "}" Linha: 29 Coluna: 10
Token: < SMB_CBC, "}" Linha: 30 Coluna: 6
```

Tabela de Símbolos de Erros (nenhum encontrado).

```
Erros

Symbol Table
1 KW program
2 KW else
3 KW if
4 KW while
5 KW write
6 KW read
7 KW num
8 KW char
9 KW not
10 KW or
11 KW and
12 ID verificaMode
13 ID destino
14 ID mode
15 ID node
16 ID link
17 ID setLink
18 ID linkrecv
19 ID getLink
```

Programa 2 Correto:

```
program_2.pasc x
* 1 program pasCTeste{
* 2     /* Programa para testar a linguagem
* 3     PasC do professor de Compiladores.
* 4
* 5     Ressalto que essa funcao e apenas para testar
* 6     nada mais do que testar.*/
* 7
* 8
* 9     char logger = "start time: "
*10     char parsed = rspec;
*11     char var = True;
*12     num list = 0 // Inicia a lista
*13     if parsed == var {
*14         logger = "start time: "
*15         list = list + 1;
*16
*17     } else {
*18         list = list - 1
*19     }
*20
*21     while (list != -1) {
*22         write "Analisando";
*23
*24     }
*25
*26     if(list>10){
*27         while(parsed==var){ write "passando"} // Imprime
*28
*29     }else{
*30         num usrp = 100
*31         if(usrp not list){
*32             write "fim"
*33         }
*34     }
*35
*36 }
```



## Tokens

```
Token: < KW, "program" Linha: 1 Coluna: 8
Token: < ID, "pasCTeste" Linha: 1 Coluna: 18
Token: < SMB_OBC, "{" Linha: 1 Coluna: 19
Token: < KW, "char" Linha: 9 Coluna: 9
Token: < ID, "logger" Linha: 9 Coluna: 16
Token: < OP_ASS, "=" Linha: 9 Coluna: 18
Token: < LIT, "start time " Linha: 9 Coluna: 33
Token: < KW, "char" Linha: 10 Coluna: 9
Token: < ID, "parsed" Linha: 10 Coluna: 16
Token: < OP_ASS, "=" Linha: 10 Coluna: 18
Token: < ID, "rspec" Linha: 10 Coluna: 24
Token: < SMB_SEM, ";" Linha: 10 Coluna: 25
Token: < KW, "char" Linha: 11 Coluna: 9
Token: < ID, "var" Linha: 11 Coluna: 13
Token: < OP_ASS, "=" Linha: 11 Coluna: 15
Token: < ID, "True" Linha: 11 Coluna: 20
Token: < SMB_SEM, ";" Linha: 11 Coluna: 21
Token: < KW, "num" Linha: 12 Coluna: 8
Token: < ID, "list" Linha: 12 Coluna: 13
Token: < OP_ASS, "=" Linha: 12 Coluna: 15
Token: < CON_NUM, "0" Linha: 12 Coluna: 17
Token: < KW, "if" Linha: 13 Coluna: 7
Token: < ID, "parsed" Linha: 13 Coluna: 14
Token: < OP_EQ, "==" Linha: 13 Coluna: 17
Token: < ID, "var" Linha: 13 Coluna: 21
Token: < SMB_OBC, "{" Linha: 13 Coluna: 23
Token: < ID, "logger" Linha: 14 Coluna: 11
Token: < OP_ASS, "=" Linha: 14 Coluna: 13
Token: < LIT, "start time " Linha: 14 Coluna: 28
Token: < ID, "list" Linha: 15 Coluna: 9
Token: < OP_ASS, "=" Linha: 15 Coluna: 11
Token: < ID, "list" Linha: 15 Coluna: 16
Token: < OP_AD, "+" Linha: 15 Coluna: 18
Token: < CON_NUM, "1" Linha: 15 Coluna: 20
Token: < SMB_SEM, ";" Linha: 15 Coluna: 21
Token: < SMB_CBC, "}" Linha: 17 Coluna: 6
Token: < KW, "else" Linha: 17 Coluna: 11
Token: < SMB_OBC, "{" Linha: 17 Coluna: 13
Token: < ID, "list" Linha: 18 Coluna: 13
Token: < OP_ASS, "=" Linha: 18 Coluna: 15
Token: < ID, "list" Linha: 18 Coluna: 20
Token: < OP_MIN, "-" Linha: 18 Coluna: 22
Token: < CON_NUM, "1" Linha: 18 Coluna: 24
Token: < SMB_CBC, "}" Linha: 19 Coluna: 6
Token: < KW, "while" Linha: 21 Coluna: 10
Token: < SMB_OPA, "(" Linha: 21 Coluna: 12
Token: < ID, "list" Linha: 21 Coluna: 16
Token: < OP_NE, "!=" Linha: 21 Coluna: 19
Token: < OP_MIN, "-" Linha: 21 Coluna: 21
Token: < CON_NUM, "1" Linha: 21 Coluna: 22
Token: < SMB_CPA, ")" Linha: 21 Coluna: 23
Token: < SMB_OBC, "{" Linha: 21 Coluna: 25
Token: < KW, "write" Linha: 22 Coluna: 14
Token: < LIT, "Analizando" Linha: 22 Coluna: 27
Token: < SMB_SEM, ";" Linha: 22 Coluna: 28
Token: < SMB_CBC, "}" Linha: 24 Coluna: 6
Token: < KW, "if" Linha: 26 Coluna: 7
```

```
Token: < SMB_OPA, "(" Linha: 26 Coluna: 8
Token: < ID, "list" Linha: 26 Coluna: 12
Token: < OP_GT, ">" Linha: 26 Coluna: 13
Token: < CON_NUM, "10" Linha: 26 Coluna: 15
Token: < SMB_CPA, ")" Linha: 26 Coluna: 16
Token: < SMB_OBC, "{" Linha: 26 Coluna: 17
Token: < KW, "while" Linha: 27 Coluna: 14
Token: < SMB_OPA, "(" Linha: 27 Coluna: 15
Token: < ID, "parsed" Linha: 27 Coluna: 21
Token: < OP_EQ, "==" Linha: 27 Coluna: 23
Token: < ID, "var" Linha: 27 Coluna: 26
Token: < SMB_CPA, ")" Linha: 27 Coluna: 27
Token: < SMB_OBC, "{" Linha: 27 Coluna: 28
Token: < KW, "write" Linha: 27 Coluna: 34
Token: < LIT, "passando" Linha: 27 Coluna: 45
Token: < SMB_CBC, "}" Linha: 27 Coluna: 46
Token: < SMB_CBC, "}" Linha: 29 Coluna: 6
Token: < KW, "else" Linha: 29 Coluna: 10
Token: < SMB_OBC, "{" Linha: 29 Coluna: 11
Token: < KW, "num" Linha: 30 Coluna: 12
Token: < ID, "usrp" Linha: 30 Coluna: 17
Token: < OP_ASS, "=" Linha: 30 Coluna: 19
Token: < CON_NUM, "100" Linha: 30 Coluna: 23
Token: < KW, "if" Linha: 31 Coluna: 11
Token: < SMB_OPA, "(" Linha: 31 Coluna: 12
Token: < ID, "usrp" Linha: 31 Coluna: 16
Token: < KW, "not" Linha: 31 Coluna: 20
Token: < ID, "list" Linha: 31 Coluna: 25
Token: < SMB_CPA, ")" Linha: 31 Coluna: 26
Token: < SMB_OBC, "{" Linha: 31 Coluna: 27
Token: < KW, "write" Linha: 32 Coluna: 18
Token: < LIT, "fim" Linha: 32 Coluna: 24
Token: < SMB_CBC, "}" Linha: 33 Coluna: 10
Token: < SMB_CBC, "}" Linha: 34 Coluna: 6
Token: < SMB_CBC, "}" Linha: 36 Coluna: 2
```

Tabela de Símbolos e Erros (nenhum encontrado).

```
Erros

Symbol Table
1 KW program
2 KW else
3 KW if
4 KW while
5 KW write
6 KW read
7 KW num
8 KW char
9 KW not
10 KW or
11 KW and
12 ID pasCTeste
13 ID logger
14 ID parsed
15 ID rspec
16 ID var
17 ID True
18 ID list
19 ID usrp
```



### Programa 3 Correto:

```
★ 1  program teste3{
★ 2      char valor;
★ 3      valor = 6 + 3;
★ 4      valor = 10 / 2;
★ 5      valor = 1.55;
★ 6
★ 7      if ( valor < 2 and valor > 50 ) {
★ 8          write valor;
★ 9      } else {
★10          valor = 5-3-4+8+2/4+35.500 // Teste 'a'
★11      }
★12      while(valor != 20){
★13          /* Fica verificando ate que
★14             valor seja igual a 20;
★15             se for diferente, imprime na tela */
★16
★17             write valor;
★18
★19             valor = " ates2434 ";
★20
★21      }
★22 }
```

## Tokens

```
Token: < KW, "program" Linha: 1 Coluna: 8
Token: < ID, "teste3" Linha: 1 Coluna: 15
Token: < SMB_OBC, "{" Linha: 1 Coluna: 16
Token: < KW, "char" Linha: 2 Coluna: 9
Token: < ID, "valor" Linha: 2 Coluna: 15
Token: < SMB_SEM, ";" Linha: 2 Coluna: 16
Token: < ID, "valor" Linha: 3 Coluna: 10
Token: < OP_ASS, "=" Linha: 3 Coluna: 13
Token: < CON_NUM, "6" Linha: 3 Coluna: 15
Token: < OP_AD, "+" Linha: 3 Coluna: 17
Token: < CON_NUM, "3" Linha: 3 Coluna: 19
Token: < SMB_SEM, ";" Linha: 3 Coluna: 20
Token: < ID, "valor" Linha: 4 Coluna: 10
Token: < OP_ASS, "=" Linha: 4 Coluna: 12
Token: < CON_NUM, "10" Linha: 4 Coluna: 15
Token: < OP_DIV, "/" Linha: 4 Coluna: 18
Token: < CON_NUM, "2" Linha: 4 Coluna: 20
Token: < SMB_SEM, ";" Linha: 4 Coluna: 21
Token: < ID, "valor" Linha: 5 Coluna: 10
Token: < OP_ASS, "=" Linha: 5 Coluna: 12
Token: < CON_NUM, "1.55" Linha: 5 Coluna: 17
Token: < SMB_SEM, ";" Linha: 5 Coluna: 18
Token: < KW, "if" Linha: 7 Coluna: 7
Token: < SMB_OPA, "(" Linha: 7 Coluna: 9
Token: < ID, "valor" Linha: 7 Coluna: 15
Token: < OP_LT, "<" Linha: 7 Coluna: 18
Token: < CON_NUM, "2" Linha: 7 Coluna: 20
Token: < KW, "and" Linha: 7 Coluna: 24
Token: < ID, "valor" Linha: 7 Coluna: 30
Token: < OP_GT, ">" Linha: 7 Coluna: 32
Token: < CON_NUM, "50" Linha: 7 Coluna: 35
Token: < SMB_CPA, ")" Linha: 7 Coluna: 37
Token: < SMB_OBC, "{" Linha: 7 Coluna: 39
Token: < KW, "write" Linha: 8 Coluna: 14
Token: < ID, "valor" Linha: 8 Coluna: 20
Token: < SMB_SEM, ";" Linha: 8 Coluna: 21
Token: < SMB_CBC, "}" Linha: 9 Coluna: 6
Token: < KW, "else" Linha: 9 Coluna: 11
Token: < SMB_OBC, "{" Linha: 9 Coluna: 13
Token: < ID, "valor" Linha: 10 Coluna: 14
Token: < OP_ASS, "=" Linha: 10 Coluna: 16
Token: < CON_NUM, "5" Linha: 10 Coluna: 18
Token: < OP_MIN, "-" Linha: 10 Coluna: 19
Token: < CON_NUM, "3" Linha: 10 Coluna: 20
Token: < OP_MIN, "-" Linha: 10 Coluna: 21
Token: < CON_NUM, "4" Linha: 10 Coluna: 22
Token: < OP_AD, "+" Linha: 10 Coluna: 23
Token: < CON_NUM, "8" Linha: 10 Coluna: 24
Token: < OP_AD, "+" Linha: 10 Coluna: 25
Token: < CON_NUM, "2" Linha: 10 Coluna: 26
Token: < OP_DIV, "/" Linha: 10 Coluna: 28
Token: < CON_NUM, "4" Linha: 10 Coluna: 29
Token: < OP_AD, "+" Linha: 10 Coluna: 30
Token: < CON_NUM, "35.500" Linha: 10 Coluna: 36
Token: < SMB_CBC, "}" Linha: 11 Coluna: 6
Token: < KW, "while" Linha: 12 Coluna: 10
Token: < SMB_OPA, "(" Linha: 12 Coluna: 11
Token: < ID, "valor" Linha: 12 Coluna: 16
Token: < OP_NE, "!=" Linha: 12 Coluna: 19
Token: < CON_NUM, "20" Linha: 12 Coluna: 22
Token: < SMB_CPA, ")" Linha: 12 Coluna: 23
Token: < SMB_OBC, "{" Linha: 12 Coluna: 24
Token: < KW, "write" Linha: 17 Coluna: 14
Token: < ID, "valor" Linha: 17 Coluna: 20
Token: < SMB_SEM, ";" Linha: 17 Coluna: 21
Token: < ID, "valor" Linha: 19 Coluna: 14
Token: < OP_ASS, "=" Linha: 19 Coluna: 16
Token: < LIT, "ates2434 " Linha: 19 Coluna: 29
Token: < SMB_SEM, ";" Linha: 19 Coluna: 30
Token: < SMB_CBC, "}" Linha: 21 Coluna: 6
Token: < SMB_CBC, "}" Linha: 22 Coluna: 2
```

Tabela de Símbolos e Erros (nenhum encontrado).

```
Erros

Symbol Table
1 KW program
2 KW else
3 KW if
4 KW while
5 KW write
6 KW read
7 KW num
8 KW char
9 KW not
10 KW or
11 KW and
12 ID teste3
13 ID valor
```

### Programa com Erro 1

Este programa teve como objetivo testar o modo pânico entendido em sala de aula.

Na linha 8 coluna 23 existe um erro de formação de *digit*. Após detectar o erro o programa continua lendo os demais caracteres até encontrar um inteiro, o que de fato estava esperando receber. Isso acontece quando encontra o 0 após a palavra negado, na linha 10 coluna 26. Posteriormente é encontrado outro erro, na linha 10 coluna 27, quando se tenta formar um literal, na qual deve-se ser feito entre aspas duplas e não pode ter quebra de linha. Existe ainda o erro para se formar constante char, na linha 3, coluna 24.

```
program_errol.pasc x
* 1  program Provision {
* 2      char logger = "Called provision in test";
* 3      char slice = 'urns';
* 4      char privileges = "True";
* 5      num user = 0
* 6      if(privileges == "True") {
* 7          write "Acesso correto";
* 8          user = user + 1.p;
* 9      } else {
*10      write "Acesso negado0"
*11      user = user -1;
*12
*13      }
*14  }
```

## Tokens

```
Token: < KW, "program" Linha: 1 Coluna: 8
Token: < ID, "Provision" Linha: 1 Coluna: 18
Token: < SMB_OBC, "{" Linha: 1 Coluna: 20
Token: < KW, "char" Linha: 2 Coluna: 9
Token: < ID, "logger" Linha: 2 Coluna: 16
Token: < OP_ASS, "=" Linha: 2 Coluna: 18
Token: < LIT, "Called provision in test" Linha: 2 Coluna: 45
Token: < SMB_SEM, ";" Linha: 2 Coluna: 46
Token: < KW, "char" Linha: 3 Coluna: 9
Token: < ID, "slice" Linha: 3 Coluna: 15
Token: < OP_ASS, "=" Linha: 3 Coluna: 17
Token: < SMB_SEM, ";" Linha: 3 Coluna: 25
Token: < KW, "char" Linha: 4 Coluna: 9
Token: < ID, "previleges" Linha: 4 Coluna: 20
Token: < OP_ASS, "=" Linha: 4 Coluna: 22
Token: < LIT, "True" Linha: 4 Coluna: 29
Token: < SMB_SEM, ";" Linha: 4 Coluna: 30
Token: < KW, "num" Linha: 5 Coluna: 8
Token: < ID, "user" Linha: 5 Coluna: 13
Token: < OP_ASS, "=" Linha: 5 Coluna: 15
Token: < CON_NUM, "0" Linha: 5 Coluna: 17
Token: < KW, "if" Linha: 6 Coluna: 7
Token: < SMB_OPA, "(" Linha: 6 Coluna: 8
Token: < ID, "previleges" Linha: 6 Coluna: 18
Token: < OP_EQ, "==" Linha: 6 Coluna: 21
Token: < LIT, "True" Linha: 6 Coluna: 28
Token: < SMB_CPA, ")" Linha: 6 Coluna: 29
Token: < SMB_OBC, "{" Linha: 6 Coluna: 31
Token: < KW, "write" Linha: 7 Coluna: 14
Token: < LIT, "Acesso correto" Linha: 7 Coluna: 31
Token: < SMB_SEM, ";" Linha: 7 Coluna: 32
Token: < ID, "user" Linha: 8 Coluna: 13
Token: < OP_ASS, "=" Linha: 8 Coluna: 15
Token: < ID, "user" Linha: 8 Coluna: 20
Token: < OP_AD, "+" Linha: 8 Coluna: 22
Token: < CON_NUM, "1.0" Linha: 10 Coluna: 23
```



## Erros

```
Erros
literal must contain only one character Linha: 3 Coluna: 24
Error encountered. Expected an integer Linha: 8 Coluna: 26
Error encountered. Expected an integer Linha: 8 Coluna: 27
Error encountered. Expected an integer Linha: 9 Coluna: 1
Error encountered. Expected an integer Linha: 9 Coluna: 5
Error encountered. Expected an integer Linha: 9 Coluna: 6
Error encountered. Expected an integer Linha: 9 Coluna: 7
Error encountered. Expected an integer Linha: 9 Coluna: 8
Error encountered. Expected an integer Linha: 9 Coluna: 9
Error encountered. Expected an integer Linha: 9 Coluna: 10
Error encountered. Expected an integer Linha: 9 Coluna: 11
Error encountered. Expected an integer Linha: 9 Coluna: 12
Error encountered. Expected an integer Linha: 9 Coluna: 13
Error encountered. Expected an integer Linha: 10 Coluna: 1
Error encountered. Expected an integer Linha: 10 Coluna: 5
Error encountered. Expected an integer Linha: 10 Coluna: 6
Error encountered. Expected an integer Linha: 10 Coluna: 7
Error encountered. Expected an integer Linha: 10 Coluna: 8
Error encountered. Expected an integer Linha: 10 Coluna: 9
Error encountered. Expected an integer Linha: 10 Coluna: 10
Error encountered. Expected an integer Linha: 10 Coluna: 11
Error encountered. Expected an integer Linha: 10 Coluna: 12
Error encountered. Expected an integer Linha: 10 Coluna: 13
Error encountered. Expected an integer Linha: 10 Coluna: 14
Error encountered. Expected an integer Linha: 10 Coluna: 15
Error encountered. Expected an integer Linha: 10 Coluna: 16
Error encountered. Expected an integer Linha: 10 Coluna: 17
Error encountered. Expected an integer Linha: 10 Coluna: 18
Error encountered. Expected an integer Linha: 10 Coluna: 19
Error encountered. Expected an integer Linha: 10 Coluna: 20
Error encountered. Expected an integer Linha: 10 Coluna: 21
Error encountered. Expected an integer Linha: 10 Coluna: 22
Error encountered. Expected an integer Linha: 10 Coluna: 23
Error encountered. Expected an integer Linha: 10 Coluna: 24
Error encountered. Expected an integer Linha: 10 Coluna: 25
Can not have line break : Linha: 11 Coluna: 1
Can not have line break : Linha: 12 Coluna: 1
Can not have line break : Linha: 13 Coluna: 1
Can not have line break : Linha: 14 Coluna: 1
Invalid Character expected " Linha: 14 Coluna: 3
```

## Tabela de Símbolo

```
Symbol Table
1 KW program
2 KW else
3 KW if
4 KW while
5 KW write
6 KW read
7 KW num
8 KW char
9 KW not
10 KW or
11 KW and
12 ID Provision
13 ID logger
14 ID slice
15 ID privileges
16 ID user
```

## Programa Erro 2

Neste programa testou-se o modo pânico na formação do operador != . Quando não se encontra o sinal de igual após o !, na linha 5 coluna 34, o analisador léxico ignora os demais caracteres, até encontrar o sinal esperado ( = ), o que acontece na linha 10 coluna 15.

```
program_erro2.pasc x
★ 1  program testeErro2{
★ 2      num valor = 6;
★ 3      num valoradmin = 10;
★ 4      num valoruser = 20;
★ 5      IF( valoruser < 9 or valor1 ! 20){
★ 6          while(){
★ 7              //calcular testar
★ 8          };
★ 9      }
★10      if(valor == 10){
★11
★12      }
★13      valor1 = valor1 - 6;
★14
★15  }
```

Tokens:

```
Token: < KW, "program" Linha: 1 Coluna: 8
Token: < ID, "testeErro2" Linha: 1 Coluna: 19
Token: < SMB_OBC, "{" Linha: 1 Coluna: 20
Token: < KW, "num" Linha: 2 Coluna: 8
Token: < ID, "valor" Linha: 2 Coluna: 14
Token: < OP_ASS, "=" Linha: 2 Coluna: 16
Token: < CON_NUM, "6" Linha: 2 Coluna: 18
Token: < SMB_SEM, ";" Linha: 2 Coluna: 19
Token: < KW, "num" Linha: 3 Coluna: 8
Token: < ID, "valoradmin" Linha: 3 Coluna: 19
Token: < OP_ASS, "=" Linha: 3 Coluna: 21
Token: < CON_NUM, "10" Linha: 3 Coluna: 24
Token: < SMB_SEM, ";" Linha: 3 Coluna: 25
Token: < KW, "num" Linha: 4 Coluna: 8
Token: < ID, "valoruser" Linha: 4 Coluna: 18
Token: < OP_ASS, "=" Linha: 4 Coluna: 20
Token: < CON_NUM, "20" Linha: 4 Coluna: 23
Token: < SMB_SEM, ";" Linha: 4 Coluna: 24
Token: < KW, "if" Linha: 5 Coluna: 7
Token: < SMB_OPA, "(" Linha: 5 Coluna: 8
Token: < ID, "valoruser" Linha: 5 Coluna: 18
Token: < OP_LT, "<" Linha: 5 Coluna: 21
Token: < CON_NUM, "9" Linha: 5 Coluna: 23
Token: < KW, "or" Linha: 5 Coluna: 26
Token: < ID, "valor1" Linha: 5 Coluna: 33
Token: < OP_NE, "!=" Linha: 10 Coluna: 15
Token: < OP_ASS, "=" Linha: 10 Coluna: 16
Token: < CON_NUM, "10" Linha: 10 Coluna: 19
Token: < SMB_CPA, ")" Linha: 10 Coluna: 20
Token: < SMB_OBC, "{" Linha: 10 Coluna: 21
Token: < SMB_CBC, "}" Linha: 12 Coluna: 6
Token: < ID, "valor1" Linha: 13 Coluna: 11
Token: < OP_ASS, "=" Linha: 13 Coluna: 13
Token: < ID, "valor1" Linha: 13 Coluna: 20
Token: < OP_MIN, "-" Linha: 13 Coluna: 22
Token: < CON_NUM, "6" Linha: 13 Coluna: 24
Token: < SMB_SEM, ";" Linha: 13 Coluna: 25
Token: < SMB_CBC, "}" Linha: 15 Coluna: 2
```

Erros:

[illegible]



## Tabela de Símbolo

```
Symbol Table
1 KW program
2 KW else
3 KW if
4 KW while
5 KW write
6 KW read
7 KW num
8 KW char
9 KW not
10 KW or
11 KW and
12 ID testeErro2
13 ID valor
14 ID valoradmin
15 ID valoruser
16 ID valor1
```

## Programa Erro 3

Com este programa visa testar a formação de *digit*, construção de literal e comentário de várias linhas.

```
program_erro3.pasc x
★ 1  program erro2{
★ 2      valor = 5;
★ 3      if (valor < 5) {
★ 4          valor = 10 * 5;
★ 5          valor = 10.o;
★ 6
★ 7      } else {
★ 8          valor = 5;
★ 9      }
★10
★11      while(valor > 5) {
★12          /* vamos testar o comentario de varias linhas.
★13             para ver se funciona mesmo */
★14      }
★15
★16      user = 'marcos'
★17
★18      if(user == "marcos"){
★19          write "Marcos"
★20      }else{
★21          /* inicia o teste dos comentarios de varias linhas
★22             na qual deve-se ignorar ate o fim de arquivo
★23      }else if(valor1 != ' teste'){
★24
★25      else(valor == 'a'){
★26
★27      }
★28
★29  }
```

## Tokens

```
Token: < KW, "program" Linha: 1 Coluna: 8
Token: < ID, "erro2" Linha: 1 Coluna: 14
Token: < SMB_OBC, "{" Linha: 1 Coluna: 15
Token: < ID, "valor" Linha: 2 Coluna: 10
Token: < OP_ASS, "=" Linha: 2 Coluna: 12
Token: < CON_NUM, "5" Linha: 2 Coluna: 14
Token: < SMB_SEM, ";" Linha: 2 Coluna: 15
Token: < KW, "if" Linha: 3 Coluna: 7
Token: < SMB_OPA, "(" Linha: 3 Coluna: 9
Token: < ID, "valor" Linha: 3 Coluna: 14
Token: < OP_LT, "<" Linha: 3 Coluna: 17
Token: < CON_NUM, "5" Linha: 3 Coluna: 19
Token: < SMB_CPA, ")" Linha: 3 Coluna: 20
Token: < SMB_OBC, "{" Linha: 3 Coluna: 22
Token: < ID, "valor" Linha: 4 Coluna: 14
Token: < OP_ASS, "=" Linha: 4 Coluna: 16
Token: < CON_NUM, "10" Linha: 4 Coluna: 19
Token: < OP_MUL, "*" Linha: 4 Coluna: 22
Token: < CON_NUM, "5" Linha: 4 Coluna: 24
Token: < SMB_SEM, ";" Linha: 4 Coluna: 25
Token: < ID, "valor" Linha: 5 Coluna: 14
Token: < OP_ASS, "=" Linha: 5 Coluna: 16
Token: < CON_NUM, "10.5" Linha: 8 Coluna: 18
Token: < SMB_SEM, ";" Linha: 8 Coluna: 19
Token: < SMB_CBC, "}" Linha: 9 Coluna: 6
Token: < KW, "while" Linha: 11 Coluna: 10
Token: < SMB_OPA, "(" Linha: 11 Coluna: 11
Token: < ID, "valor" Linha: 11 Coluna: 16
Token: < OP_GT, ">" Linha: 11 Coluna: 18
Token: < CON_NUM, "5" Linha: 11 Coluna: 20
Token: < SMB_CPA, ")" Linha: 11 Coluna: 21
Token: < SMB_OBC, "{" Linha: 11 Coluna: 23
Token: < SMB_CBC, "}" Linha: 14 Coluna: 6
Token: < ID, "user" Linha: 16 Coluna: 9
Token: < OP_ASS, "=" Linha: 16 Coluna: 11
Token: < KW, "if" Linha: 18 Coluna: 7
Token: < SMB_OPA, "(" Linha: 18 Coluna: 8
Token: < ID, "user" Linha: 18 Coluna: 12
Token: < OP_EQ, "==" Linha: 18 Coluna: 15
Token: < LIT, "marcos" Linha: 18 Coluna: 24
Token: < SMB_CPA, ")" Linha: 18 Coluna: 25
Token: < SMB_OBC, "{" Linha: 18 Coluna: 26
Token: < KW, "write" Linha: 19 Coluna: 10
Token: < LIT, "Marcos" Linha: 19 Coluna: 19
Token: < SMB_CBC, "}" Linha: 20 Coluna: 6
Token: < KW, "else" Linha: 20 Coluna: 10
Token: < SMB_OBC, "{" Linha: 20 Coluna: 11
```

## Erros e Tabela de Símbolo

```
Erros
Error encountered. Expected an integer Linha: 5 Coluna: 21
Error encountered. Expected an integer Linha: 5 Coluna: 22
Error encountered. Expected an integer Linha: 6 Coluna: 1
Error encountered. Expected an integer Linha: 7 Coluna: 1
Error encountered. Expected an integer Linha: 7 Coluna: 5
Error encountered. Expected an integer Linha: 7 Coluna: 6
Error encountered. Expected an integer Linha: 7 Coluna: 7
Error encountered. Expected an integer Linha: 7 Coluna: 8
Error encountered. Expected an integer Linha: 7 Coluna: 9
Error encountered. Expected an integer Linha: 7 Coluna: 10
Error encountered. Expected an integer Linha: 7 Coluna: 11
Error encountered. Expected an integer Linha: 7 Coluna: 12
Error encountered. Expected an integer Linha: 7 Coluna: 13
Error encountered. Expected an integer Linha: 8 Coluna: 1
Error encountered. Expected an integer Linha: 8 Coluna: 5
Error encountered. Expected an integer Linha: 8 Coluna: 9
Error encountered. Expected an integer Linha: 8 Coluna: 10
Error encountered. Expected an integer Linha: 8 Coluna: 11
Error encountered. Expected an integer Linha: 8 Coluna: 12
Error encountered. Expected an integer Linha: 8 Coluna: 13
Error encountered. Expected an integer Linha: 8 Coluna: 14
Error encountered. Expected an integer Linha: 8 Coluna: 15
Error encountered. Expected an integer Linha: 8 Coluna: 16
Error encountered. Expected an integer Linha: 8 Coluna: 17
literal must contain only one character Linha: 16 Coluna: 20
Invalid Character expected */ Linha: 29 Coluna: 3
```