**Átomos reconhecidos pelo mini analisador léxico**

**Identificador**: começado por uma letra e seguido por dígitos ou letras.

**Número inteiro**: começado digito e seguido mais dígitos;

**Operador de atribuição**: caractere “:=”

**Palavra reservada while**: caracteres “while”

**arquivo de entrada SEM ERRO**

|  |  |
| --- | --- |
| 1  2  3 | var1 1 :=  124 while 567 While  Var1:= 123 |

saída do analisador léxico:

linha: 1 – identificador – var1

linha: 1 – numero\_inteiro – 1

linha: 1 - operador\_atribuicao –

linha: 2 – numero\_inteiro – 124

linha: 2 – while –

linha: 2 – numero\_inteiro – 567

linha: 2 – while –

linha: 3 – identificador – var1

linha: 3 – operador\_atribuicao –

linha: 3 – numero\_inteiro – 123

linha: 3 – fim de string(EOS)

**arquivo de entrada COM ERRO**

|  |  |
| --- | --- |
| 1  2  3 | variavel 123te  100a  := while 21b |

saída do analisador léxico:

linha: 1 – identificador – variavel

linha: 1 – ERRO –

void testaAnalisadorLexico(){

.......

while( true ){

TInfoAtomo info\_atomo = obter\_atomo();

......

if( info\_atomo.atomo == EOS ){

// fim de analise léxica

break;

}

}

}

typdef enum{

ERRO,

IDENTIFICADOR,

NUMERO\_INTEIRO,

ATRIBUICAO,

WHILE,

EOS

}TAtomo;

typedef struct{

  TAtomo atomo;

  int linha;

  int atributo\_numero;

  char atributo\_ID[15];

}TInfoAtomo;

TInfoAtomo obter\_atomo(){

// ignora os caracteres delimitadores

// (espaços em branco, tabulação, nova linha e retorno de carro)

// lembre-se que o fim de buffer gera o átomo EOS.

TInfoAtomo infoAtomo;

infoAtomo.atomo = ERRO;

if( isdigit(\*buffer)){

//chama função que reconhece digito

return reconhece\_digito();

if( isalpha(\*buffer)){

return reconhece\_id();

.............

else

return infoAtomo;

TInfoAtomo reconhece\_digito(){

TinfoAtomo atoma\_num;

átomo\_num.atomo = NUMERO\_INTEIRO;

átomo\_num. atributo\_numero = valor convertido