Lexer

Tabela de Cadeias

Nome	Padrão	Válida	Inválida
KEYWORD	if|else|while|for|int|decimal|string|void|null|return|true|false	if, else, while	ifif, fi, _if, if_
ID	[a-zA-Z_][a-zA-Z0-9_]*	x, _n_2, A2, _123_, __	1x, .x2, 1_a
DECIMAL	[0-9]+\.[0-9]+	0.0, 00.00, 13.97, 09.030	0e2.2, -3.3, 76
INT	[0-9]+	0, 00, 1397, 09030	0e2, -3, 76.2
STRING	""([^\\""]|\\.)*""	“”, “\””, “hello \world\!”	“, “\”, “hello \world!\”
ASSIGN	=	Literal	Tudo além do literal
EQUAL	==	Literal	Tudo além do literal
NOT_EQUAL	!=	Literal	Tudo além do literal
LARGER	>	Literal	Tudo além do literal
LARGER_EQUAL	>=	Literal	Tudo além do literal
SMALLER	<	Literal	Tudo além do literal
SMALLER_EQUAL	<=	Literal	Tudo além do literal
ADD	+	Literal	Tudo além do literal

Nome	Padrão	Válida	Inválida
SUBTRACT	-	Literal	Tudo além do literal
DIVIDE	/	Literal	Tudo além do literal
MULTIPLY	*	Literal	Tudo além do literal
AND	&	Literal	Tudo além do literal
OR	|	Literal	Tudo além do literal
NOT	!	Literal	Tudo além do literal
LEFT_PARENTHESIS	(	Literal	Tudo além do literal
RIGHT_PARENTHESIS	)	Literal	Tudo além do literal
LEFT_BRACE	{	Literal	Tudo além do literal
RIGHT_BRACE	}	Literal	Tudo além do literal
SEMICOLON	;	Literal	Tudo além do literal
EMPTY	[\t\r]	Espaços, tabs, e carriage return	Tudo além de espaços em branco
NEW_LINE	\n	Nova linha	Tudo além da nova linha

Testes

Palavras-chave e IDs

```
Lexer:
        1 - Read from shell
        2 - Read from file
        3 - Exit
1
Enter code to analyze:
if else while for int decimal string void null return true false
KEYWORD   ->  if
KEYWORD   ->  else
KEYWORD   ->  while
KEYWORD   ->  for
KEYWORD   ->  int
KEYWORD   ->  decimal
KEYWORD   ->  string
KEYWORD   ->  void
KEYWORD   ->  null
KEYWORD   ->  return
KEYWORD   ->  true
KEYWORD   ->  false
```

```
Enter code to analyze:
if _if ifif aif ifa if_ _1if if1 if123 if_1
KEYWORD   ->  if
ID   ->  _if
ID   ->  ifif
ID   ->  aif
ID   ->  ifa
ID   ->  if_
ID   ->  _1if
ID   ->  if1
ID   ->  if123
ID   ->  if_1
```

```
Enter code to analyze:
_asd __ _12 asd123 AA_123_AA_123___1 12A_A
ID   ->  _asd
ID   ->  __
ID   ->  _12
ID   ->  asd123
ID   ->  AA_123_AA_123___1
INT   ->  12
ID   ->  A_A
```

Strings, ints e decimais

```
Enter code to analyze:
"" "\"" "123 123 123 asdf a3 42 \ \\\ \ \ \ \\\ asd \ \" "
STRING   ->  ""
STRING   ->  "\""
STRING   ->  "123 123 123 asdf a3 42 \ \\\ \ \ \ \\\ asd \ \" "
```

```
Enter code to analyze:
123456789 000 010 01203 0000000
INT    ->  123456789
INT    ->  000
INT    ->  010
INT    ->  01203
INT    ->  0000000
```

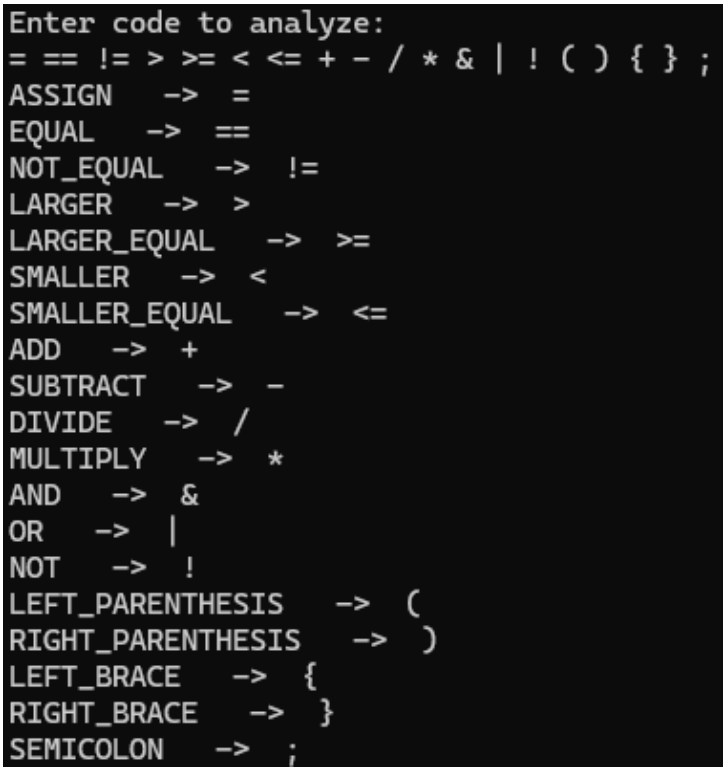
```
Enter code to analyze:
603498.124135 0000.00000 0.12319059 00023.320000
DECIMAL    ->  603498.124135
DECIMAL    ->  0000.00000
DECIMAL    ->  0.12319059
DECIMAL    ->  00023.320000
```

```
Enter code to analyze:
"" "
STRING   ->  ""

Lexer Error on file <stdin>: Unterminated string literal
    Start:
        Line: 1
        Column: 4
    End:
        Line: 1
        Column: 5
```

```
Enter code to analyze:
"\"" "\"
STRING   ->  "\""

Lexer Error on file <stdin>: Unterminated string literal
    Start:
        Line: 1
        Column: 6
    End:
        Line: 1
        Column: 8
```

```
Enter code to analyze:
" asd asd sad as

Lexer Error on file <stdin>: Unterminated string literal
    Start:
        Line: 1
        Column: 1
    End:
        Line: 1
        Column: 17
```

Operadores e Delimitadores



```
Enter code to analyze:
!!====>>=<<====+-/*&|(){};
NOT   ->  !
NOT_EQUAL   ->  !=
EQUAL   ->  ==
ASSIGN   ->  =
LARGER   ->  >
LARGER_EQUAL   ->  >=
SMALLER   ->  <
SMALLER_EQUAL   ->  <=
EQUAL   ->  ==
ASSIGN   ->  =
ADD   ->  +
SUBTRACT   ->  -
DIVIDE   ->  /
MULTIPLY   ->  *
AND   ->  &
OR   ->  |
LEFT_PARENTHESIS   ->  (
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```

Geral

```
Enter code to analyze:
decimal _x2 = 0.23; _x2 = _x2 / 12; if (_x2 < 0.01) { while (_x2 > 0) { _x2 = _x2 - 0.001 }; return true; } else { return false; };
KEYWORD   ->  decimal
ID   ->  _x2
ASSIGN   ->  =
DECIMAL   ->  0.23
SEMICOLON   ->  ;
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
DIVIDE   ->  /
INT   ->  12
SEMICOLON   ->  ;
KEYWORD   ->  if
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
SMALLER   ->  <
DECIMAL   ->  0.01
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
KEYWORD   ->  while
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
LARGER   ->  >
INT   ->  0
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
SUBTRACT   ->  -
DECIMAL   ->  0.001
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
KEYWORD   ->  return
KEYWORD   ->  true
SEMICOLON   ->  ;
RIGHT_BRACE   ->  }
KEYWORD   ->  else
LEFT_BRACE   ->  {
KEYWORD   ->  return
KEYWORD   ->  false
SEMICOLON   ->  ;
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```

Arquivo

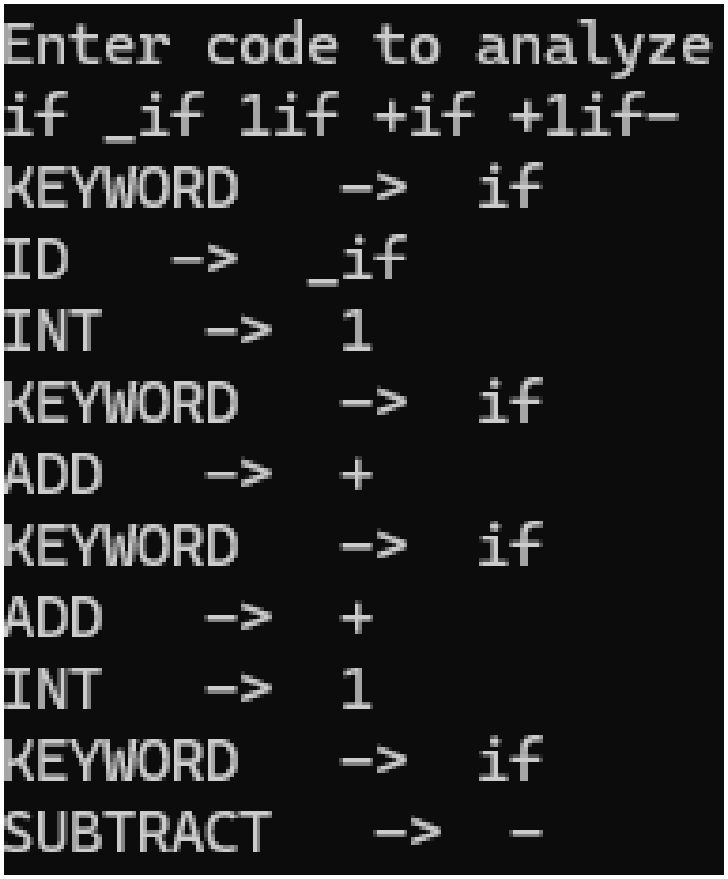
```
decimal _x2 = 0.23;

_x2 = _x2 / 12;

if (_x2 < 0.01)
{
        while (_x2 > 0)
        {
                _x2 = _x2 - 0.001
        };
        return true;
}
else
{
        return false;
};
```

```
Enter filepath:
test.txt
KEYWORD   ->  decimal
ID   ->  _x2
ASSIGN   ->  =
DECIMAL   ->  0.23
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
NEW_LINE   ->  NEW_LINE
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
DIVIDE   ->  /
INT   ->  12
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  if
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
SMALLER   ->  <
DECIMAL   ->  0.01
RIGHT_PARENTHESIS   ->  )
NEW_LINE   ->  NEW_LINE
LEFT_BRACE   ->  {
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  while
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
LARGER   ->  >
INT   ->  0
RIGHT_PARENTHESIS   ->  )
NEW_LINE   ->  NEW_LINE
LEFT_BRACE   ->  {
NEW_LINE   ->  NEW_LINE
ID   ->  _x2
ASSIGN   ->  =
```

```
ID   ->  _x2
SUBTRACT   ->  -
DECIMAL   ->  0.001
NEW_LINE   ->  NEW_LINE
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  return
KEYWORD   ->  true
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
RIGHT_BRACE   ->  }
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  else
NEW_LINE   ->  NEW_LINE
LEFT_BRACE   ->  {
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  return
KEYWORD   ->  false
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```

Palavra-Chave vs. Identificador

Como mostrado acima, no segundo teste de palavras-chave e IDs, palavras-chave como if só são reconhecidas se não forem diretamente precedidas ou sucedidas de carácter que casa com o padrão de IDs. Portanto: if = KEYWORD, qualquer outra variação com if = ID.



No teste acima vemos que somente o segundo if é ID, pois ele é precedido de underline (_), que casa com o padrão de ID.

Modularização e Estados

Todo o domínio foi criado com foco em modularização e sem repetição de código. Por exemplo, se quisessemos modificar o operador de multiplicação para #, bastaria modificar a declaração da literal de adição, que refletirá na Regex e em todos os outros pontos utilizado.

Quanto ao estado e tratamento de erros, foram criadas classes LexerError e Context. O Context possuí índice, linha, coluna e nome do arquivo, e o LexerError possui Context inicial, final, e mensagem de erro. Há um Context do Lexer, que avança conforme a leitura de caracteres, e cria cópias de si mesmo no início de tokens compostos, para corretamente indicar o início e fim de qualquer erro.

```
17 references
public class Context
{
    5 references
    public int Index { get; set; }
    5 references
    public int Line { get; set; }
    6 references
    public int Column { get; set; }
    3 references
    public string FileName { get; set; }

    2 references
    public Context(int index, int line, int column, string fileName)
    {
        Index = index;
        Line = line;
        Column = column;
        FileName = fileName;
    }

    1 reference
    public void Advance(char? current)
    {
        Index++;

        if (current is '\n')
        {
            Line++;
            Column = 0;
        } else
        {
            Column++;
        }
    }

    2 references
    public Context Copy() => new(Index, Line, Column, FileName);
}
```

```
8 references
public class LexerError : Exception
{
    4 references
    public Context Start { get; set; }
    3 references
    public Context End { get; set; }

    4 references
    public LexerError(Context start, Context end, string message) : base(message)
    {
        Start = start;
        End = end;
    }

    1 reference
    public override string ToString()
    {
        return $"""
            Lexer Error on file {Start.FileName}: {Message}
                Start:
                    Line: {Start.Line + 1}
                    Column: {Start.Column + 1}
                End:
                    Line: {End.Line + 1}
                    Column: {End.Column}
            """;
    }
}
```