**Protótipo de Linguagem “PortugolDemo”:s Analisador Léxico e Versão Inicial do Analisador Sintático com ANTLR4**

**Pedro Benitah Vieira Sanchez de Melo**

CENTRO UNIVERSITÁRIO DO PARÁ - CESUPA – ESCOLA DE NEGÓCIOS, TECNOLOGIA E INOVAÇÃO - ARGO – CURSO DE CIÊNCIA DA COMPUTAÇÃO – Belém, PA – Brasil

pedrobenitah@gmail.com

***Abstract.*** *This article presents the specification and implementation of a lexical analyzer (scanner) and an initial version of a syntactic analyzer (parser) for a subset of a Portuguese-based language called “Portugol.” The project was developed using ANTLR4 and C#. In brief, the solution comprises: definitions of lexeme classes, a token table, regular expressions, a context-free grammar (CFG) with precedence and associativity, lexical–syntactic integration, and execution evidence. The prototype supports declarations, a main block, conditionals, loops, arithmetic and logical expressions, as well as functions and procedures with return values.*

***Resumo.*** *Neste artigo, é apresentada a especificação e a implementação de um analisador léxico (scanner) e de uma versão inicial de um analisador sintático (parser) para um subconjunto de uma linguagem com termos em português chamada “Portugol”, o projeto foi desenvolvido utilizando ANTLR4 e C#. falando de maneira sucinta sobre solução, ela contempla: definição de classes de lexemas, tabela de tokens, expressões regulares, gramática livre de contexto (GLC) com precedência e associatividade, integração léxico–sintático e evidências de execução. O protótipo suporta declarações, bloco principal main, condicionais, laços, expressões aritméticas e lógicas, além de funções e procedimentos com retorno.*

***Keywords*** *— Compilers; ANTLR; Lexical Analysis; Syntax Analysis; Programming Languages.*

***Palavras‑chave*** *— Compiladores; ANTLR; Análise Léxica; Análise Sintática; Linguagens de Programação.*

**1. Introdução**

PortugolDemo é uma linguagem educacional minimalista (estilo 'Portugol') criada para demonstrar as fases de um compilador/intérprete: análise léxica, análise sintática, noções de análise semântica e execução. A implementação utiliza ANTLR4 para gerar o lexer e o parser a partir de uma gramática livre de contexto e integra tudo em C# (.NET).

Escopo desta entrega:

• Introduzir o intérprete PortugolDemo e a linguagem associada;

• Descrever aspectos técnicos e estágios do pipeline;

• Demonstrar tokens, árvore e execução, incluindo exemplos de erro.

**2. Intérprete Selecionado e Linguagem Associada**

Intérprete: PortugolDemo (implementação própria).

Conceito da linguagem: sintaxe em português, tipos primitivos (inteiro, real, logico, texto), funções com retorno, procedimentos sem retorno, bloco principal 'programa ... fimPrograma', comandos de controle (se/senao, enquanto, para), aritmética, relacionais e lógicos (e/ou/nao).

Exemplo breve:

inteiro soma(inteiro x, inteiro y) inicio retorne x + y; fim

programa main() inicio escreva("ok"); fim fimPrograma

**3. Histórico e Evolução do Projeto**

• v0.1: léxico inicial, tokens CSV e impressão de árvore (-–tokens, –-tree).

• v0.2: funções e procedimentos; precedência de expressões; if/while/for; visitor interpretativo.

• v0.3: polimento de erros, flags –-both e execução do programa de teste completo.

**4. Arquitetura Interna e Módulos**

Módulos principais:

• Lexer (PortugolLexer): tokenização a partir de expressões regulares e palavras-chave.

• Parser (PortugolParser): GLC com regras para declarações, comandos e expressões.

• Árvore/Contextos: nós gerados pelo parser (RuleContext) e visita via padrão Visitor.

• Visitor (Exec): interpretação — tabela de símbolos, escopos, chamadas, controle de fluxo e retorne.

• CLI: flags --tokens, --tree, --both, execução padrão sem flags.

Diagrama (simplificado):

Fonte → Lexer(ANTLR) → Tokens → Parser(ANTLR) → Árvore (Parse Tree) → Visitor(Exec) → Saída/Erros

**5. Estágios do Processo de Compilação/Interpretação**

**5.1 Análise Léxica (tokenização)**

Expressões regulares (amostras): ID, NUM\_LITERAL, STRING\_LITERAL, BOOL\_LITERAL; palavras-chave e operadores.

| Classe | Descrição | Regex / Exemplos |
| --- | --- | --- |
| ID | Identificadores | [a-zA-Z\_][a-zA-Z0-9\_]\* (main, soma) |
| NUM\_LITERAL | Números | \d+(\.\d+)? (10, 2.5) |
| STRING\_LITERAL | Strings | \" (\\.|[^\"\\\r\n])\* \" (\"ok\") |
| BOOL\_LITERAL | Booleanos | verdadeiro | falso |

**5.2 Análise Sintática (parsing)**

GLC com precedência: ou > e > relações > soma > termo > unário > primário. Regras para funções/procedimentos, comandos (declaração, atribuição, if/else, enquanto, para, retorne) e chamadas.

**5.3 Análise Semântica (protótipo)**

Semântica mínima implementada no visitor: tabela de símbolos, checagem de declaração antes de uso, coerção numérica simples (inteiro→real) em operações/chamadas.

**5.4 Otimizações**

Otimizações não foram aplicadas nesta fase.

**5.5 Geração de Código / Execução**

Execução interpretada diretamente sobre a árvore via visitor (sem bytecode). I/O por 'escreva', fluxo por 'se/senao', 'enquanto' e 'para'.

**6. Ferramentas e Bibliotecas**

• ANTLR4 (geração de lexer/parser; flags -visitor -listener; destino C#).

•.NET / C# (integração: AntlrInputStream, CommonTokenStream, PortugolLexer / Parser).

• PowerShell para pipelines (redirecionamento para CSV).

Comandos (exemplos):

java -jar .\tools\antlr-4.13.2-complete.jar -Dlanguage=CSharp -visitor -listener .\GrammarSpec\Portugol.g4 -o .\Grammar

dotnet run -- --tokens .\testes\teste.por | Out-File .\out\tokens.csv -Encoding utf8

dotnet run -- --tree .\testes\teste.por

dotnet run -- .\testes\teste.por

**7. Exemplo: Estágios Exibidos e Erros**

Programa de teste (completo):

1. inteiro soma(inteiro x, inteiro y)
2. inicio
3. retorne x + y;
4. fim
5. programa main()
6. inicio
7. inteiro a = 10;
8. escreva("soma=", soma(a, 5));
9. fim
10. fimPrograma

**7.1 Cadeia de tokens (amostra)**

| Tipo | Lexema | Linha | Coluna |
| --- | --- | --- | --- |
| inteiro' | inteiro | 1 | 0 |
| ID | soma | 1 | 8 |
| (' | ( | 1 | 12 |
| inteiro' | inteiro | 1 | 13 |
| ID | x | 1 | 21 |
| COMMA | , | 1 | 22 |
| inteiro' | inteiro | 1 | 24 |
| ID | y | 1 | 32 |
| )' | ) | 1 | 33 |
| inicio' | inicio | 2 | 0 |
| retorne' | retorne | 3 | 4 |
| ID | x | 3 | 12 |
| +' | + | 3 | 14 |
| ID | y | 3 | 16 |
| ;' | ; | 3 | 17 |
| fim' | fim | 4 | 0 |
| programa' | programa | 6 | 0 |
| ID | main | 6 | 9 |
| (' | ( | 6 | 13 |
| )' | ) | 6 | 14 |
| inicio' | inicio | 7 | 0 |
| inteiro' | inteiro | 8 | 4 |
| ID | a | 8 | 12 |
| =' | #ERROR! | 8 | 14 |
| NUM\_LITERAL | 10 | 8 | 16 |
| ;' | ; | 8 | 18 |
| escreva' | escreva | 9 | 4 |
| (' | ( | 9 | 11 |
| STRING\_LITERAL | "soma=" | 9 | 12 |
| COMMA | , | 9 | 19 |
| ID | soma | 9 | 21 |
| (' | ( | 9 | 25 |
| ID | a | 9 | 26 |
| COMMA | , | 9 | 27 |
| NUM\_LITERAL | 5 | 9 | 29 |
| )' | ) | 9 | 30 |
| )' | ) | 9 | 31 |
| ;' | ; | 9 | 32 |
| fim' | fim | 10 | 0 |
| fimPrograma' | fimPrograma | 11 | 0 |

**7.2 Árvore sintática**

1. programa
2. declaracaoFuncao
3. tipo
4. inteiro
5. soma
6. (
7. listaParams
8. param
9. tipo
10. inteiro
11. x
12. ,
13. param
14. tipo
15. inteiro
16. y
17. )
18. bloco
19. inicio
20. comandos
21. comando
22. comandoRetorne
23. retorne
24. expr
25. relacao
26. soma
27. termo
28. un
29. prim
30. x
31. +
32. termo
33. un
34. prim
35. y
36. ;
37. fim
38. programa
39. main
40. (
41. )
42. blocoPrincipal
43. inicio
44. comandos
45. comando
46. declaracaoVar
47. tipo
48. inteiro
49. a
50. =
51. expr
52. relacao
53. soma
54. termo
55. un
56. prim
57. 10
58. ;
59. comando
60. chamadaProc
61. escreva
62. (
63. listaArgs
64. expr
65. relacao
66. soma
67. termo
68. un
69. prim
70. "soma="
71. ,
72. expr
73. relacao
74. soma
75. termo
76. un
77. prim
78. chamadaFunc
79. soma
80. (
81. listaArgs
82. expr
83. relacao
84. soma
85. termo
86. un
87. prim
88. a
89. ,
90. expr
91. relacao
92. soma
93. termo
94. un
95. prim
96. 5
97. )
98. )
99. ;
100. fim
101. fimPrograma
102. <EOF>

**7.3 Execução e mensagens de erro (demonstrações)**

* Erro léxico:

1. programa main() inicio
2. escreva(10 @ 5);
3. fim fimPrograma

Saída (esperada):

1. Erros LÉXICO:
2. [linha 2, coluna 15] token recognition error at: '@'

* Erro sintático:

1. programa main() inicio
2. se (verdadeiro entao
3. inicio fim
4. fim fimPrograma

Saída (esperada):

1. Erros SINTÁTICO:
2. [linha 2, coluna 20] perto de 'entao': mismatched input 'entao' expecting ')'

* Erro semântico:

1. programa main() inicio
2. escreva(z);
3. fim fimPrograma

Saída (esperada no visitor):

1. Falha durante a análise:
2. variavel nao declarada: z

**8. Resultados (execução bem-sucedida)**

Saída obtida na execução do arquivo de teste completo:

1. == ====Teste==== ==
2. condicao 1: ok
3. while i=0, ehPar=verdadeiro
4. while i=1, ehPar=falso
5. while i=2, ehPar=verdadeiro
6. for i=0
7. for i=1
8. for i=2
9. a=10
10. b=2.5
11. c=30
12. d=2.8
13. flag=verdadeiro
14. msg=Teste
15. a=10, b=2.5, c=30, d=2.8, flag=verdadeiro, msg=Teste

**8. Conclusão**

O PortugolDemo atende aos itens solicitados: apresentação do intérprete e linguagem, estágios do pipeline, artefatos gerados (tokens/árvore) e execução com exemplos de erro. Como próximos passos: checagem de tipos, suporte a arrays/records, melhoria das mensagens de erro e geração de código intermediário.

**Referências**

T. Parr, The Definitive ANTLR 4 Reference, Pragmatic Bookshelf.

Documentação .NET e C#, Microsoft Docs, 2025.

Site oficial ANTLR4 (gramáticas e destino C#).