# **Relatório Técnico: Desenvolvimento de Compilador Simples em C# com ANTLR4**

## **1. Objetivo do Projeto**

O projeto tem como objetivo criar um **compilador simples**, escrito em **C#** e executado no **Visual Studio Code**, utilizando a **ferramenta ANTLR4** para realizar o processo de **análise léxica e sintática**. A linguagem alvo é propositalmente simples, utilizando comandos como:

escreva("Olá, mundo!");

## **2. Tecnologias Utilizadas**

* **.NET SDK 9.0**
* **C# como linguagem de implementação**
* **ANTLR 4.13.0** (com suporte à geração de código em C#)
* **VS Code como ambiente de desenvolvimento**
* **Extensão .NET CLI para gerenciamento do projeto**

## **3. Estrutura da Linguagem**

A linguagem foi definida por meio de uma **gramática ANTLR** no arquivo Linguagem.g4, com a seguinte definição:

grammar Linguagem;

programa: comando+ ;

comando: escreva ;

escreva: 'escreva' '(' TEXTO ')' ';' ;

TEXTO: '"' (~["\r\n])\* '"' ;

WS: [ \t\r\n]+ -> skip ;

### **Explicação:**

* A linguagem permite apenas comandos escreva("texto");.
* TEXTO é definido como qualquer conteúdo entre aspas duplas (exceto aspas ou quebras de linha).
* WS ignora espaços em branco, tabulações e quebras de linha.

## **4. Integração com ANTLR4**

A gramática foi compilada com o seguinte comando:

java -jar tools/antlr-4.13.0-complete.jar -Dlanguage=CSharp -o Generated Gramatica/Linguagem.g4

Esse comando gerou os seguintes arquivos na pasta Generated:

* LinguagemLexer.cs
* LinguagemParser.cs
* LinguagemListener.cs
* LinguagemBaseListener.cs

Esses arquivos foram incluídos no projeto C# via \*.csproj ou compilados automaticamente com base na estrutura do projeto.

## **5. Interpretador da Linguagem**

Foi criada uma classe Interpretador que herda de LinguagemBaseListener, responsável por implementar a lógica de execução para o comando escreva.

### **Código da Classe:**

using System;

public class Interpretador : LinguagemBaseListener

{

public override void EnterEscreva(LinguagemParser.EscrevaContext context)

{

var texto = context.TEXTO().GetText();

Console.OutputEncoding = System.Text.Encoding.UTF8; // Corrige exibição de acentos

Console.WriteLine(texto);

}

}

## **6. Problemas Encontrados e Soluções**

### **6.1. Erro de Reconhecimento de Tokens**

Entrada inválida como Olá, teste gerava erros:

token recognition error at: 'O'

token recognition error at: 'l'

...

**Motivo:** A gramática esperava a palavra-chave escreva, seguida de um texto entre aspas. Qualquer entrada fora desse padrão era inválida.

### **6.2. Erro de Parsing**

Entrada incompleta como escreva causava:

mismatched input '<EOF>' expecting '('

**Motivo:** O parser esperava um parêntese após escreva, conforme a gramática.

### **6.3. NullReferenceException**

Quando o texto esperado não era encontrado, a função context.TEXTO() retornava null.

**Solução:** Adicionada verificação no código do interpretador para evitar exceções ao acessar objetos nulos.

### **6.4. Problema com Caracteres Especiais**

Mesmo com a entrada correta, a saída aparecia como:

"Ol├í, mundo!"

**Motivo:** O console não estava configurado para UTF-8.

**Soluções Aplicadas:**

* Inserção do comando Console.OutputEncoding = System.Text.Encoding.UTF8;.
* Instrução para executar chcp 65001 no terminal.

Após isso, a entrada correta como:

escreva("Olá, mundo!");

Passou a gerar a saída esperada:

"Olá, mundo!"

## **7. Execução e Teste**

No método Main do projeto, a execução básica envolve:

var input = Console.ReadLine();

var inputStream = new AntlrInputStream(input);

var lexer = new LinguagemLexer(inputStream);

var tokens = new CommonTokenStream(lexer);

var parser = new LinguagemParser(tokens);

var tree = parser.programa();

var listener = new Interpretador();

ParseTreeWalker.Default.Walk(listener, tree);

Esse código:

* Lê uma linha de entrada do usuário.
* Processa com o lexer e o parser.
* Caminha pela árvore de sintaxe com um interpretador personalizado.

## **8. Considerações Finais**

O compilador está funcionando corretamente para a linguagem simples definida, com suporte ao comando escreva. A estrutura modular permite fácil expansão, como a adição futura de comandos leia, estruturas condicionais, expressões matemáticas, etc.

Problemas iniciais como codificação de caracteres e exceções em tempo de execução foram resolvidos com configurações apropriadas de console e validação de entrada no interpretador.