



UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Trabalho Prático - CCF 441

Piccola Lingua Brasiliana

Aroldo Augusto Barbosa Simões - 4250

Fabiano Lara Leroy - 4227

Igor Nascimento Silva - 4257

Juan Pablo Andrade Avelar - 4229

Matheus Silva Palhares - 4249

Florestal - MG

2025

Sumário

1. Introdução	3
3. Desenvolvimento	5
3.1 Definições da linguagem	6
3.2 Makefile	8
4. Resultados	9
5. Conclusão	10
6. Referências	11

1. Introdução

Esse documento corresponde à documentação do Trabalho Prático da disciplina CCF 441 - Compiladores, ministrada pelo professor Daniel Mendes Barbosa na Universidade Federal de Viçosa – Campus Florestal. O principal objetivo deste trabalho é proporcionar aos alunos a vivência prática no desenvolvimento de do *frontend* de um compilador, por meio da criação de uma linguagem de programação original e da implementação incremental de suas fases: análise léxica, análise sintática, análise semântica e geração de código. A proposta busca não apenas consolidar os conhecimentos teóricos estudados ao longo da disciplina, mas também estimular a criatividade, o trabalho em equipe e o uso de ferramentas como Flex (LEX) e Yacc (Bison), fundamentais no processo de construção de compiladores reais.

Com base nas orientações fornecidas na especificação fornecida pelo professor, o trabalho prático da disciplina está subdividido em três etapas consecutivas e complementares. Na primeira parte (trabalho prático 1), o grupo deve criar a especificação completa de sua própria linguagem de programação. Isso inclui definir o nome da linguagem, sua origem e contexto criativo, os tipos de dados primitivos, comandos disponíveis, palavras-chave e reservadas, além da gramática inicial com suas variáveis, terminais (tokens) e padrões léxicos. É essencial que a linguagem seja procedural e que sua sintaxe traga criatividade, indo além de simples traduções de linguagens existentes. Após essa definição, o grupo deve implementar um analisador léxico funcional, utilizando Flex (LEX) com a linguagem C, capaz de identificar e imprimir os tokens reconhecidos a partir de arquivos fonte da linguagem. A entrega deve conter o arquivo .l, exemplos de uso, e um PDF com a documentação explicativa do funcionamento e das decisões tomadas.

Já na segunda parte(trabalho prático 2), deve-se dar continuidade ao desenvolvimento com a criação do analisador sintático da linguagem. Utilizando o Yacc (Bison) em conjunto com o Flex, o grupo deve implementar um compilador que, ao receber um programa-fonte como entrada, imprima o código com linhas numeradas, a tabela de símbolos construída, e informe se o programa é sintaticamente correto ou incorreto, com localização e descrição de possíveis erros. É esperado que o analisador léxico seja adaptado para integrar-se ao parser e que erros léxicos e sintáticos sejam detectados e reportados. A documentação

atualizada deve incluir a nova gramática (caso modificada), exemplos de entradas válidas e inválidas, e instruções de compilação e uso do compilador.

Por fim, na terceira parte (trabalho prático 3), o grupo deve realizar a análise semântica, como a verificação de tipos, e implementar a geração de código. Essa geração pode ser feita por meio de representações intermediárias como código de três endereços, árvores sintáticas abstratas ou até mesmo IR da LLVM. É permitido escolher a forma mais adequada conforme os objetivos do grupo, sendo possível também gerar código para simuladores ou plataformas específicas. A entrega final deve conter o código-fonte completo, os arquivos de teste, e um PDF com a documentação detalhando como a análise semântica foi feita, o funcionamento da geração de código, exemplos representativos, e quais construções da linguagem estão cobertas. Esta etapa finaliza o ciclo de desenvolvimento do *frontend* do compilador.

Na primeira parte do trabalho o grupo desenvolveu *Piccola Lingua Brasiliana*, fortemente inspirada no meme contemporâneo Italian Brainrot. A linguagem utiliza personagens fictícios caricatos, com base nos personagens criados por IA do meme, derivados dos nomes dos integrantes do grupo e pessoas reais, por exemplo "Fabianito Innamorato" e "Danieluzzo Supremo", para representar comandos e estruturas da linguagem de forma cômica e original.

A linguagem foi idealizada a partir do meme conhecido como Italian Brainrot, um fenômeno humorístico da internet que satiriza a forma como o idioma italiano é percebido de forma caricata por falantes de outras línguas. O meme consiste em inventar palavras com sonoridade similar ao italiano real, geralmente compostas por repetições, terminações vocálicas e ritmo musical, como "*bambalino*", "*pastarini*", ou frases absurdas como "*mamma mia spagetto di compilatore*".

Essa estética exagerada, associada ao estereótipo do gestual expansivo e apaixonado atribuído aos italianos, viralizou em redes sociais como Twitter e TikTok, sendo frequentemente usada em piadas visuais e vídeos curtos. Inspirados por esse universo cômico e sonoro, criamos uma linguagem de programação cujas palavras-chave evocam esse estilo — combinando humor, criatividade e um desafio técnico de construção léxica e sintática. A proposta oferece, assim, um ambiente lúdico de aprendizado e experimentação na construção de compiladores, explorando conceitos formais em um cenário culturalmente inventivo.

2. Organização

O projeto foi organizado em um repositório estruturado para facilitar o desenvolvimento e a manutenção do compilador da linguagem *Piccola Lingua Brasileira*. A divisão em pastas separa claramente os arquivos de código-fonte, documentação, entradas de teste e saídas geradas, garantindo um fluxo de trabalho eficiente. O uso de ferramentas como LEX e makefile automatiza processos essenciais, como a geração do analisador léxico e a compilação do projeto, enquanto arquivos como entrada.txt permitem testes rápidos e validam a funcionalidade do sistema.

A documentação detalhada, incluindo o arquivo principal em PDF e guias auxiliares, assegura que todas as etapas do projeto estejam bem registradas, desde a especificação da linguagem até a implementação prática. Essa abordagem organizada não apenas facilita o trabalho em equipe, mas também prepara o terreno para as próximas fases do compilador, como a análise sintática e semântica, mantendo a clareza e a consistência em todo o processo.

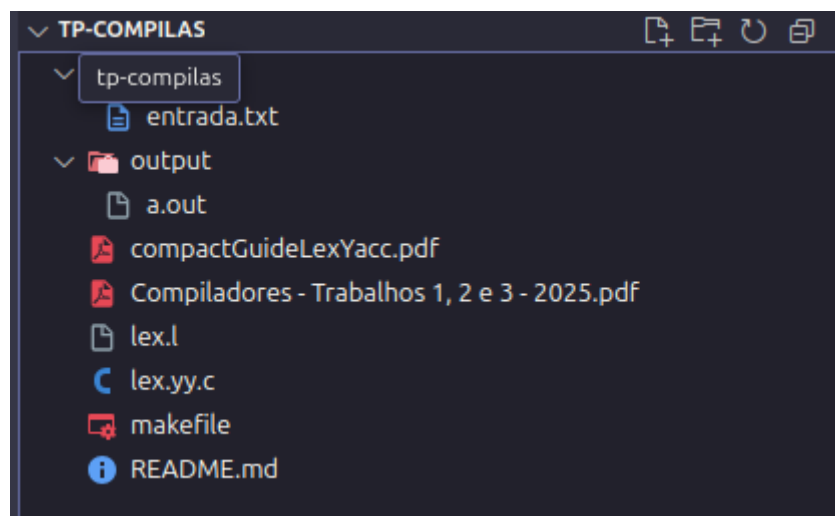


Figura 1 - Repositório do projeto.

3. Desenvolvimento

A linguagem *Piccola Lingua Brasileira* surgiu a partir da proposta de unir criatividade, identidade cultural e humor como elementos centrais do projeto. O

nome, que pode ser traduzido livremente como “Pequena Linguagem Brasileira” em um italiano improvisado, remete de forma cômica e propositalmente exagerada ao meme conhecido como Italian Brainrot — uma sátira da forma como o italiano é representado exageradamente na internet, com palavras inventadas e expressões que soam “italianas” mesmo não tendo base na língua real.

Inspirando-se nesse meme, a linguagem adota palavras-chave que imitam a fonética do italiano por meio de aliterações, repetições sonoras e terminações típicas, como “TralaleroTralala”, “BombardinoCrocodilo” e “LaVacaSaturnoSaturnita”. A intenção é provocar uma experiência leve, divertida e lúdica tanto para quem lê quanto para quem escreve programas nessa linguagem, ao mesmo tempo em que se explora conceitos sérios da construção de linguagens formais, como análise léxica e sintática.

Além disso, a linguagem segue características fundamentais para um compilador didático: possui tipagem estática, exige indentação para definição de blocos (em sintonia com a ideia de código indentado) e diferencia claramente palavras-chave de identificadores — por isso, reforça-se que “palavra reservada é palavra-chave”, estabelecendo as estruturas fixas do idioma. Todas as palavras reservadas da linguagem iniciam com letra maiúscula para melhor legibilidade.

O resultado é uma linguagem que, mesmo caricata em sua superfície, foi cuidadosamente projetada para abranger os principais elementos necessários para a construção de um compilador: tipos primitivos, comandos de controle de fluxo, operadores aritméticos e delimitação de blocos e expressões. Tudo isso em um universo estético próprio, baseado no humor e na irreverência de um meme internacional adaptado ao contexto brasileiro.

3.1 Definições da linguagem

Valor padrão	Significado	<i>Piccola Lingua Brasiliana</i>
int	Tipo primitivo inteiro	TralaleroTralala
string	Tipo primitivo que representa texto	BombardinoCrocodilo
boolean	Tipo primitivo que representa valores lógicos	BrbrbrParapim

for	Comando para laços de repetição	ChipampanziniBananini
while	Comando para laços de repetição	BallerinaCappuccina
{	Delimitam blocos de código	BallerinaGlaucinna
}	Delimitam blocos de código	GlaucinnaBallerina
(Delimitam blocos de código	Aprire
)	Delimitam blocos de código	Chiudere
=	Operador de atribuição	TungTungTungAroldo
+	Operador de adição	BanditoCarioquito
-	Operador de subtração	BombardinoFabaiano
*	Operador de multiplicação	BombouliniBoulinis
/	Operador de divisão	ChocofantoElefanto
/*	Abertura de comentário	NoooLa
*/	Fechamento de comentário	PoliziaNooo
const	Declaração de constantes	RhinoToasterino
import	Importador de módulos/bibliotecas	Importare
function	Definição de função	LaVacaSaturnoSaturnita

Tabela 1 - Definições da linguagem

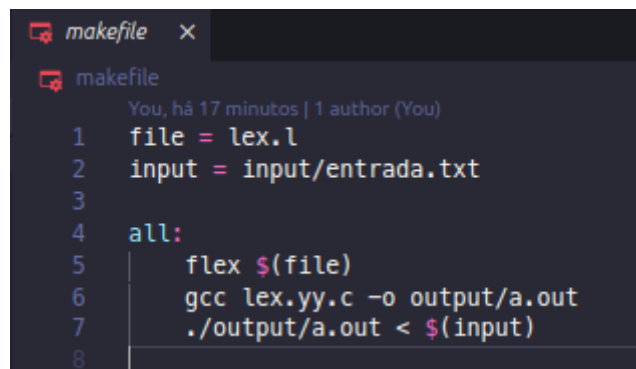
Número	<i>Piccola Lingua Brasiliana</i>
0	Zero
1	Uno
2	Due
3	Tre
4	Quattro
5	Cinque
6	Sei

7	Sette
8	Otto
9	Nove

Tabela 2 - Tabela de valores numéricos da linguagem

3.2 Makefile

O arquivo makefile apresentado tem como objetivo automatizar o processo de compilação e execução do analisador léxico desenvolvido para a linguagem criada. Nele, são definidas variáveis como 'file' (que aponta para o arquivo de especificação léxica, lex.l) e 'input' (que indica o arquivo de entrada contendo o código-fonte a ser analisado, input/entrada.txt). A regra principal, 'all', executa três comandos sequenciais: primeiro, o Flex é utilizado para gerar o código C (lex.yy.c) a partir do arquivo léxico; em seguida, o GCC compila esse código, produzindo um executável (output/a.out); por fim, o executável é rodado, processando o arquivo de entrada e exibindo os tokens e eventuais erros léxicos. Essa automatização simplifica o fluxo de trabalho, garantindo consistência e eficiência durante o desenvolvimento e testes do compilador.



```

makefile x
makefile
You, há 17 minutos | 1 author (You)
1 file = lex.l
2 input = input/entrada.txt
3
4 all:
5     flex $(file)
6     gcc lex.yy.c -o output/a.out
7     ./output/a.out < $(input)
8

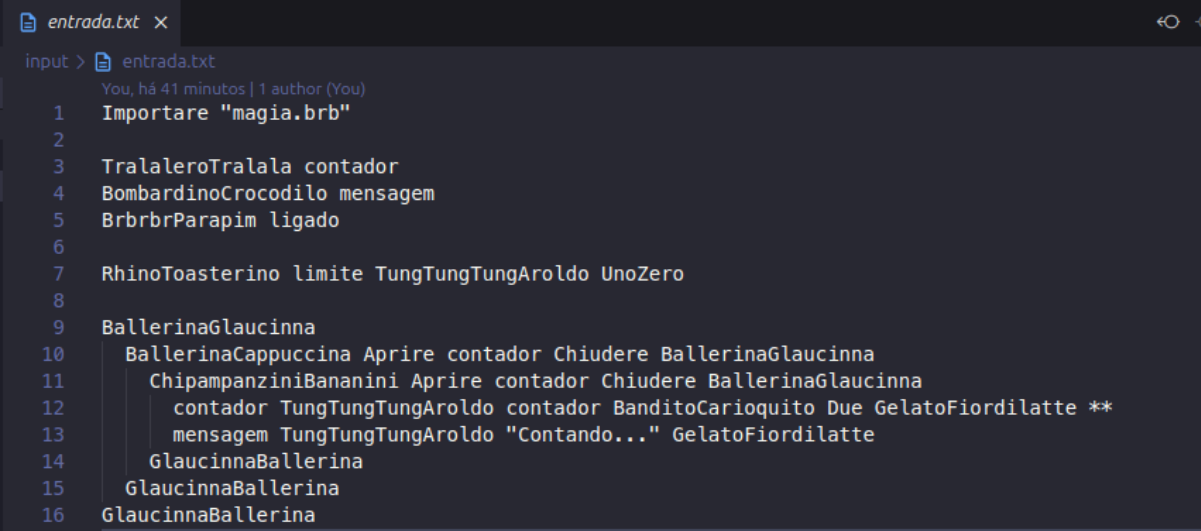
```

Figura 2 - Arquivo Makefile

4. Resultados

O analisador léxico desenvolvido para a linguagem *Piccola Lingua Brasiliana* demonstrou eficiência na identificação e classificação dos tokens definidos na especificação da linguagem. Durante os testes realizados, o analisador foi capaz de processar corretamente o arquivo de entrada (entrada.txt), reconhecendo

palavras-chave, identificadores, literais, operadores e delimitadores conforme esperado.



```
input > entrada.txt
You, há 41 minutos | 1 author (You)
1 Importare "magia.brb"
2
3 TralaleroTralala contador
4 BombarinoCrocodilo mensagem
5 BrbrbrParapim ligado
6
7 RhinoToasterino limite TungTungTungAroldo UnoZero
8
9 BallerinaGlaucinna
10 BallerinaCappuccina Aprire contador Chiudere BallerinaGlaucinna
11 ChipampanziniBananini Aprire contador Chiudere BallerinaGlaucinna
12 contador TungTungTungAroldo contador BanditoCarioquito Due GelatoFiordilatte **
13 mensagem TungTungTungAroldo "Contando..." GelatoFiordilatte
14 GlaucinnaBallerina
15 GlaucinnaBallerina
16 GlaucinnaBallerina
```

Figura 4 - Entrada

Alguns exemplos de tokens reconhecidos incluem:

- TOKEN_IMPORT: Importare para a palavra-chave de importação.
- TOKEN_TIPO_INT: TralaleroTralala para o tipo primitivo inteiro.
- TOKEN_ATRIBUICAO: TungTungTungAroldo para o operador de atribuição.
- TOKEN_WHILE: BallerinaCappuccina para a estrutura de repetição while.
- Número reconhecido: Unozero para o numeral "10" em formato estilizado.

Além disso, o analisador identificou e reportou um erro léxico, como símbolo inválido não definido na gramática da linguagem. Por exemplo:

- ERRO LÉXICO: Símbolo inválido: * Foi detectado quando um caractere não reconhecido foi encontrado no código-fonte.

A saída gerada pelo analisador léxico foi organizada e legível, facilitando a verificação manual e a correção de possíveis inconsistências no código-fonte testado. O uso do arquivo makefile automatizou o processo de compilação e execução, garantindo eficiência durante os testes.

Em resumo, os resultados obtidos validaram a implementação do analisador léxico, cumprindo os requisitos da primeira etapa do trabalho e preparando o terreno para as próximas fases de análise sintática e semântica. A documentação e os testes realizados asseguram a robustez da solução desenvolvida.

```

> make
flex lex.l
lex.l:31: warning, aplicação da regra não gerou nenhum resultado
gcc lex.yy.c -o output/a.out
./output/a.out < input/entrada.txt
TOKEN_IMPORT: Importare
TOKEN_STRING_LITERAL: "magia.brb"
TOKEN_TIPO_INT: TralaleroTralala
TOKEN_IDENTIFICADOR: contador
TOKEN_TIPO_STRING: BombardinoCrocodilo
TOKEN_IDENTIFICADOR: mensagem
TOKEN_TIPO_BOOLEAN: BrbrbrParapim
TOKEN_IDENTIFICADOR: ligado
TOKEN_CONST: RhinoToasterino
TOKEN_IDENTIFICADOR: limite
TOKEN_ATRIBUICAO: TungTungTungAroldo
Número reconhecido: UnoZero
TOKEN_ABRE_BLOCO: BallerinaGlaucinna
TOKEN_WHILE: BallerinaCappuccina
TOKEN_ABRE_PARENTESSES: Aprire
TOKEN_IDENTIFICADOR: contador
TOKEN_FECHA_PARENTESSES: Chiudere
TOKEN_ABRE_BLOCO: BallerinaGlaucinna
TOKEN_FOR: ChipampanziniBananini
TOKEN_ABRE_PARENTESSES: Aprire
TOKEN_IDENTIFICADOR: contador
TOKEN_FECHA_PARENTESSES: Chiudere
TOKEN_ABRE_BLOCO: BallerinaGlaucinna
TOKEN_IDENTIFICADOR: contador
TOKEN_ATRIBUICAO: TungTungTungAroldo
TOKEN_IDENTIFICADOR: contador
TOKEN_ADICAO: BanditoCarioquito
Número reconhecido: Due
TOKEN_IDENTIFICADOR: GelatoFiordilatte
ERRO LÉXICO: Símbolo inválido: *
ERRO LÉXICO: Símbolo inválido: *
TOKEN_IDENTIFICADOR: mensagem
TOKEN_ATRIBUICAO: TungTungTungAroldo
TOKEN_STRING_LITERAL: "Contando..."
TOKEN_IDENTIFICADOR: GelatoFiordilatte
TOKEN_FECHA_BLOCO: GlaucinnaBallerina
TOKEN_FECHA_BLOCO: GlaucinnaBallerina
TOKEN_FECHA_BLOCO: GlaucinnaBallerina

```

Figura 4 - Resultados

5. Conclusão

O desenvolvimento do analisador léxico para a linguagem *Piccola Lingua Brasileira* representou um marco fundamental no projeto de construção do compilador, cumprindo com êxito os objetivos da primeira etapa do trabalho prático. A implementação, baseada em Flex (LEX), demonstrou eficácia ao reconhecer tokens, identificar erros léxicos e validar a estrutura básica da linguagem, conforme evidenciado pelos testes realizados. A abordagem criativa adotada — inspirada no

meme Italian Brainrot — não apenas enriqueceu o aspecto lúdico do projeto, mas também reforçou a importância da originalidade na definição de linguagens de programação, alinhando teoria e prática de forma inovadora.

A organização do código-fonte, a documentação detalhada e a automação via makefile facilitaram a reprodutibilidade e a manutenção do sistema, enquanto os resultados obtidos (como a detecção de símbolos inválidos e a classificação precisa de tokens) comprovaram a robustez da solução. Os desafios encontrados, como ajustes na sintaxe do makefile e a adaptação de padrões léxicos, foram superados com colaboração em equipe e revisão iterativa, consolidando habilidades essenciais para as próximas etapas.

Com a base léxica estabelecida, o grupo está preparado para avançar rumo à análise sintática (Trabalho Prático 2), onde a integração com Yacc (Bison) e a expansão da gramática permitirão verificar a estrutura hierárquica do código-fonte. Este projeto não apenas atende aos requisitos acadêmicos, mas também evidencia o potencial da criatividade no campo da computação, transformando um conceito humorístico em uma ferramenta pedagógica funcional. A continuidade do trabalho promete aprofundar o entendimento sobre compiladores e reforçar a aplicação de técnicas formais em um contexto dinâmico e colaborativo.

6. Referências

1. Github. Disponível em: <<https://github.com/Palhares17/tp-compilas>>. Último acesso em: 16 de maio de 2025.
2. namu.wiki. Italian Brainrot/등장 캐릭터. Disponível em: <<https://en.namu.wiki/w/Italian%20Brainrot/등장%20캐릭터>>. Último acesso em: 16 de maio de 2025.
3. DeepSeek Chat. Modelo de IA para geração de texto e auxílio em projetos acadêmicos. Desenvolvido pela DeepSeek. Disponível em: <<https://www.deepseek.com>>. Último acesso em: 16 de maio de 2025.
4. Aho, A. V.; Lam, M. S.; Sethi, R.; Ullman, J. D. Compiladores: Princípios, Técnicas e Ferramentas. 2ª ed. Pearson, 2008. Disponível em: <<https://www.pearson.com>>. Último acesso em: 16 de maio de 2025.

5. Levine, J. flex & bison: Text Processing Tools. O'Reilly Media, 2009. Disponível em: <<https://www.oreilly.com>>. Último acesso em: 16 de maio de 2025.
6. LLVM Project. The LLVM Compiler Infrastructure. Disponível em: <<https://llvm.org/>>. Último acesso em: 16 de maio de 2025.
7. JFLAP. Software para ensino e aprendizado de teoria da computação. Disponível em: <<http://www.jflap.org/>>. Último acesso em: 16 de maio de 2025.
8. Barbosa, D. M. Material didático da disciplina de Compiladores. Universidade Federal de Viçosa – Campus UFV-Florestal, 2025. Último acesso em: 16 de maio de 2025.
9. Especificação do trabalho prático. Disponível em: <[Compiladores - Trabalhos 1, 2 e 3 - 2025](#)>. Último acesso em: 16 de maio de 2025.