Universidade Anhembi Morumbi – Mooca

MINI COMPILADOR EM JAVA

Nome: Cauã Máximo Oliveira

RA: 12523138074

Curso: Ciências da Computação

Unidade Curricular: Teoria da Computação e Compiladores

Professor: Wellington Fernando

São Paulo

2025

Introdução

Compiladores são ferramentas fundamentais no desenvolvimento de software, pois traduzem programas escritos por humanos para uma forma compreendida pelo computador. Com o objetivo de entender como esse processo ocorre internamente, este trabalho apresenta a implementação de um mini compilador utilizando a linguagem Java. A solução proposta suporta uma linguagem própria e simplificada, com instruções básicas para declaração de variáveis, expressões matemáticas e saída de dados. O projeto permite visualizar, de forma prática, o funcionamento das etapas principais da compilação, desde a análise do código-fonte até sua execução.

Objetivos

Este projeto tem como objetivo implementar um mini compilador acadêmico em Java, com foco nos seguintes pontos:

- Compreender e aplicar as etapas essenciais do processo de compilação;
- Implementar análise léxica para reconhecimento de tokens;
- Desenvolver análise sintática com base em uma gramática definida;
- Construir uma árvore sintática abstrata representando o programa;
- Executar as instruções do código-fonte através de um interpretador.

Além disso, busca-se fortalecer o aprendizado prático sobre o funcionamento interno de compiladores e a criação de pequenas linguagens de programação.

Descrição da Linguagem Criada

A linguagem desenvolvida neste projeto foi criada com o objetivo de demonstrar o funcionamento essencial de um compilador/interpretador de forma simples e funcional. Ela permite a manipulação de variáveis inteiras, execução de operações aritméticas básicas e exibição de resultados na tela.

A estrutura da linguagem é minimalista: cada instrução deve ser escrita em uma única linha e o comando `print` é utilizado para saída de dados. A linguagem também aceita comentários utilizando `//`, permitindo que o programador documente o código sem interferir na execução.

As expressões matemáticas podem incluir soma (+), subtração (-), multiplicação (*) e divisão (/), sempre avaliadas da esquerda para a direita, sem prioridade de operadores. Variáveis são criadas no momento da atribuição, mediante o uso do operador =. Caso uma variável seja utilizada antes de ser atribuída, o interpretador identifica e exibe um erro semântico. Também há verificação de erros como comandos inválidos e divisão por zero, garantindo maior robustez ao processamento.

Essa linguagem cumpre seu propósito acadêmico ao ilustrar conceitos fundamentais de construção de um interpretador: análise simples de comandos, armazenamento de variáveis, avaliação de expressões e tratamento de erros.

- **Declaração de variáveis inteiras** utilizando a palavra-chave int
- **Atribuição de valores** a variáveis previamente declaradas
- **Execução de expressões aritméticas** com os operadores +, -, * e /
- **Exibição de resultados na tela** por meio do comando print(...)

Todas as instruções devem **terminar com ponto e vírgula (;)**, o que indica ao compilador o final de um comando.

Regras gerais da linguagem

- Variáveis precisam ser declaradas antes de serem utilizadas
- Os nomes de variáveis devem começar com uma letra e podem conter números
- Não há suporte para tipos diferentes de dados além de inteiros
- Não existe controle de fluxo (como if ou while) nesta versão
- Espaços em branco são ignorados pelo compilador

Arquitetura do Compilador

O compilador foi desenvolvido seguindo uma arquitetura modular, onde cada etapa é responsável por uma parte específica do processamento do código-fonte. O objetivo é transformar um programa escrito na linguagem proposta em resultados executáveis, passando por diferentes fases. As principais etapas implementadas foram: análise léxica, análise sintática, construção da árvore sintática abstrata (AST) e interpretação.

Análise Léxica

Também chamada de scanner, essa etapa recebe o código como texto puro e converte em tokens.

Os tokens são unidades básicas da linguagem, como:

- Palavras-chave (int, print)
- Identificadores (nomes de variáveis)
- Números
- Operadores (+, -, *, /, =)
- Símbolos (;, (,))

Nesta etapa já são eliminados espaços em branco. Classe responsável: **Lexer.java**

Exemplo de análise léxica:

Entrada: x = 10 + 2;

Saída:

IDENTIFIER('x')

ASSIGN('=')

NUMBER('10')

PLUS('+')

NUMBER('2')

SEMICOLON(';')

Análise Sintática

Também conhecida como parser, essa etapa recebe a lista de tokens e verifica se a estrutura do programa está correta conforme a gramática da linguagem.

Se a sintaxe está correta, é construída uma AST

Se há erro, uma mensagem é exibida ao usuário

Classe responsável: **Parser.java**

Árvore Sintática Abstrata (AST)

A AST representa o programa de forma estruturada, permitindo ao interpretador entender o significado de cada comando.

Interpretação / Execução

Ao invés de gerar código de máquina ou outra linguagem, este projeto utiliza um **interpretador** que percorre a AST e executa cada comando em memória.

Funções do interpretador:

- Mantém uma tabela de variáveis e seus valores
- Calcula o valor de expressões
- Gera a saída no console para instruções print()

Classe responsável: **Interp.java**

Conclusão

O desenvolvimento deste mini compilador proporcionou uma compreensão prática sobre o funcionamento interno de linguagens de programação, desde a análise do código-fonte até a interpretação das instruções. A criação de uma linguagem simples permitiu aplicar conceitos fundamentais como tokenização, análise sintática e execução orientada por regras.

Apesar de apresentar recursos básicos, o projeto cumpre seu objetivo de demonstrar as principais etapas do processo de compilação, servindo como base para futuras expansões, como suporte a variáveis, expressões matemáticas mais complexas, controle de fluxo e tratamento de erros aprimorado.

Com isso, o compilador se consolida como um exercício acadêmico relevante para o aprendizado e evolução na área de desenvolvimento de linguagens e construção de ferramentas de software mais robustas.