Compiladores

Capítulo da Aula 1: Apresentação e Iniciação

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

Fevereiro de 2026

1 Objetivos

- Compreender o funcionamento e as regras da disciplina.
- Conhecer a bibliografia e ferramentas essenciais.
- Conectar a disciplina com conceitos de Estruturas de Dados, Teoria da Computação e Linguagens de Programação.

2 Introdução

Bem-vindo ao curso de Compiladores. Esta disciplina é considerada por muitos como o "divisor de águas" no curso de Ciência da Computação, pois ela integra conhecimentos de Estruturas de Dados, Algoritmos, Arquitetura de Computadores e Teoria da Computação em um único projeto prático.

3 Estrutura do Curso

O curso é dividido em dois pilares fundamentais que caminham em paralelo:

3.1 1. Fundamentação Teórica (60%)

Estudaremos os algoritmos e estruturas matemáticas que permitem a tradução de linguagens.

- **Léxico**: Autômatos Finitos e Expressões Regulares.
- **Sintático**: Gramáticas Livres de Contexto e Autômatos de Pilha.
- **Semântico**: Sistemas de Tipos e Tabelas de Símbolos.
- **Código**: Grafos de Fluxo de Controle e Otimização.

3.2 2. Projeto Prático: O Compilador Mini-Pascal (40%)

Você não apenas estudará a teoria, mas construirá, do zero, um software capaz de ler código fonte e gerar um executável real.

- **Linguagem Fonte**: Mini-Pascal (um subconjunto estrito de Pascal).
- **Linguagem Alvo**: Java Bytecode (para rodar na JVM).
- **Ferramentas**: Java 21, ANTLR4, ASM.

4 O Projeto Prático: Mini-Pascal

O projeto é desenvolvido incrementalmente ao longo do semestre. O objetivo é construir um compilador para a linguagem **Mini-Pascal**, gerando Bytecode para a **Java Virtual Machine (JVM)**. As etapas são:

- **Etapa 1: Infraestrutura e AST**: Definição da hierarquia de classes que representará o código em memória (Árvore Sintática Abstrata).
- **Etapa 2: Analisador Léxico (Manual)**: Implementação de autômatos para quebrar o texto em tokens.
- **Etapa 3: Analisador Sintático (Manual)**: Implementação de um parser descendente recursivo.
- **Etapa 4: Ferramentas (ANTLR4)**: Substituição dos analisadores manuais por geradores profissionais.
- **Etapa 5: Semântica (Escopo)**: Validação de variáveis declaradas e resolução de nomes.
- **Etapa 6: Semântica (Tipos)**: Verificação de compatibilidade de tipos (ex: não somar booleano com inteiro).
- **Etapa 7: Geração de Código**: Tradução da AST validada para instruções da JVM usando a biblioteca ASM.

△ Importante

O projeto é **incremental**. Cada etapa depende do sucesso da anterior. O acúmulo de atrasos pode inviabilizar a entrega final.

5 Conexões Interdisciplinares

Compiladores é a disciplina onde "tudo se encaixa".

5.1 Estruturas de Dados (AED2)

O compilador é, essencialmente, um grande transformador de estruturas de dados.

- **Árvores**: A estrutura central do compilador é a **AST (Abstract Syntax Tree)**. Algoritmos de caminhamento em árvore (Depth-First Search, Post-Order Traversal) são usados extensivamente para análise semântica e geração de código.
- **Tabelas Hash**: Fundamental para a **Tabela de Símbolos**, garantindo acesso $O(1)$ para buscar variáveis pelo nome.
- **Grafos**:
  - **Grafos de Fluxo de Controle (CFG)**: Usados para otimização e análise de caminhos de execução.
  - **Grafos de Interferência**: Usados no algoritmo de Coloração de Grafos para alocação de registradores.

5.2 Linguagens de Programação

Para construir um compilador, você precisa entender profundamente como as linguagens funcionam "por baixo do capô".

- **Escopo e Visibilidade**: Como funciona o sombreamento de variáveis? (`{ int x; { int x; } }`). O compilador precisa gerenciar uma pilha de tabelas de símbolos para resolver isso corretamente.
- **Sistemas de Tipos**: A diferença entre tipagem estática (verificada pelo compilador) e dinâmica. Polimorfismo e coerção.
- **Runtime Environment**: Como a memória é organizada (Stack vs Heap) para suportar chamadas de função e alocação dinâmica.

5.3 Linguagens Formais e Autômatos

A teoria da computação fornece as bases matemáticas para que nossos algoritmos sejam corretos e eficientes.

- **Expressões Regulares e Autômatos Finitos**: São a base para a construção de Scanners eficientes. Sem eles, escreveríamos "spaghetti code" cheio de `if/else` para ler caracteres.
- **Gramáticas Livres de Contexto (GLC)**: Essenciais para especificar a sintaxe estruturada (aninhamento de blocos e expressões). O parser é, na verdade, um Autômato de Pilha determinístico.

6 Ferramentas de Trabalho

Neste curso, a familiaridade com as ferramentas é crucial:

- **Git**: Todo o trabalho será versionado. Commits atômicos e mensagens claras são esperados.
- **ANTLR4**: Uma ferramenta poderosa que, a partir de uma gramática (.g4), gera todo o código Java para ler e validar o código fonte.
- **IntelliJ/VS Code**: Use uma IDE robusta. O debug será seu melhor amigo.

7 Referências

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*.
- Cooper, K., & Torczon, L. (2011). *Engineering a Compiler*.