CLP - Conceitos de Linguagens de Programação

Aula 01: Conceitos de Linguagens de Programação

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Março de 2026

A Primeira Programadora

- A primeira linguagem de programação foi criada por **Ada Lovelace** (**Primeira programadora**).
- Ela foi assistente e amiga de **Charles Babbage**, inventor da calculadora mecânica (Máquina Analítica).
- **Você Sabia?** A linguagem de programação **ADA** foi batizada em sua homenagem.

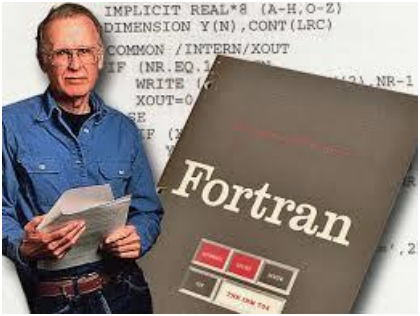
Ada Lovelace

Fortran: O Início do Alto Nível

- **Fortran** (*IBM Mathematical Formula Translation System*).
- Primeira linguagem de alto nível amplamente usada, criada na década de 1950 por **John Backus**.
- Foi criada na década de 1950.
- Continua sendo usada até hoje.
- Focada em cálculos científicos e matemáticos.

Exemplo de Código Fortran

```
IMPLICIT REAL*8 (A-H, O-Z)
DIMENSION Y(N), CONT(LRC)
```



John Backus (Fortran)

COBOL: Foco em Negócios

- **COBOL** (*Common Business Oriented Language*).
- Criada na década de 1950 com forte influência de **Grace Hopper**.
- Orientada ao processamento de bancos de dados e aplicações comerciais.
- Mantém-se ativa, integrando-se atualmente com Orientação a Objetos e .NET.

Grace Hopper

Linguagens Populares (Índice Tiobe)

Existem centenas de linguagens. Segundo o índice Tiobe, as mais populares incluem:

- Java / C / C++
- Python / C#
- JavaScript
- PHP / Swift
- Ruby / Pascal
- MATLAB / R
- Assembly
- Objective-C
- COBOL / Delphi

1ª Geração: Orientação à Máquina

- São linguagens onde suas estruturas de controle são aparentemente orientadas a máquina.
- As instruções condicionais não são aninhadas e dependem fortemente de instruções de desvio como `GOTO`.
- Uma linguagem típica desta geração é a linguagem **Fortran**.

2ª Geração: Estruturas e Nomes

- São linguagens onde suas estruturas de controle são estruturadas de forma a minimizar ou dispensar o uso do `GOTO`.
- Uma das grandes contribuições desta geração foi sua **estrutura de nomes**.
  - Permitiu melhor controle de espaços de nomes e uma eficiente alocação dinâmica de memória.
- Uma linguagem típica desta geração é o **Algol 60**.

3ª Geração: Simplicidade e Eficiência

- São linguagens que dão ênfase a simplicidade e eficiência.
  - As estruturas de controle são mais simples e eficientes.
- Uma linguagem típica desta geração é a linguagem **Pascal**.

Niklaus Wirth (Pascal)

4ª Geração: Modularização e Encapsulamento

- A maioria das linguagens desta geração focam na modularização e no encapsulamento.
- Uma linguagem típica desta geração é a linguagem **ADA**.

5ª Geração: Diversidade de Paradigmas

- Nesta geração, são agrupadas diversas linguagens com diversos paradigmas como:
  - Paradigma orientado a objetos
  - Paradigma funcional
  - Paradigma lógico

6ª Geração: Abstração Visual (Hipótese)

- Programação baseada em modelos e interfaces visuais.
- **Conceito:** O código é gerado a partir de diagramas ou especificações de alto nível.
- **Exemplo:** UML (Unified Modeling Language), Low-code/No-code tools.

7ª Geração: Assistência por I.A. (Hipótese)

- Programação através de Linguagem Natural e modelos de linguagem.
- **Foco:** Intenção do programador vs. sintaxe rígida.
- **Exemplo:** AI Coding Agents (GitHub Copilot, Cursor, etc.), LLMs (Large Language Models).

Paradigmas Imperativos

Focam em **mudanças de estado** e comandos sequenciais.

- **Procedural:** Chamadas sucessivas a procedimentos (Fortran, Basic).
- **Estrutura de Blocos:** Escopos aninhados (Algol 60, Pascal, C).
- **Orientação a Objetos:** Interação entre objetos (C++, Java, Python, Ruby).
- **Computação Distribuída:** Execução independente de rotinas (Ada).

Paradigmas Declarativos

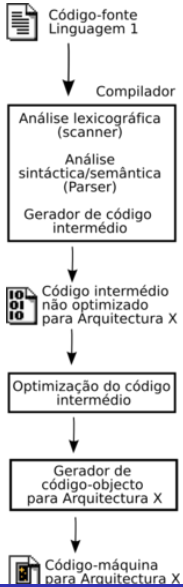
O programa especifica **o que** deve ser feito (relação ou função), não como.

- **Funcional:** Mapeamento direto entre entrada e saída através de funções (Lisp, Scheme, Haskell).
- **Lógico:** Baseado em relações e predicados (Prolog, Gödel).

Estrutura de Tipos

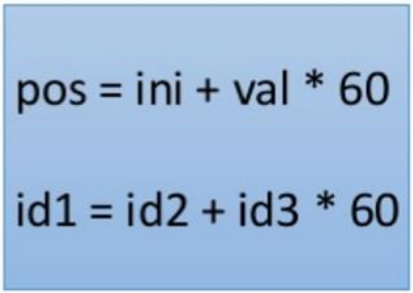
Tipo	Descrição	Exemplos
Fraca	O tipo muda dinamicamente conforme a situação.	PHP, Smalltalk
Forte	O tipo se mantém até a variável ser descartada.	Java, Ruby
Dinâmica	Definida em tempo de execução.	Python, Perl
Estática	Definida em tempo de compilação.	Java, C

Linguagens Compiladas

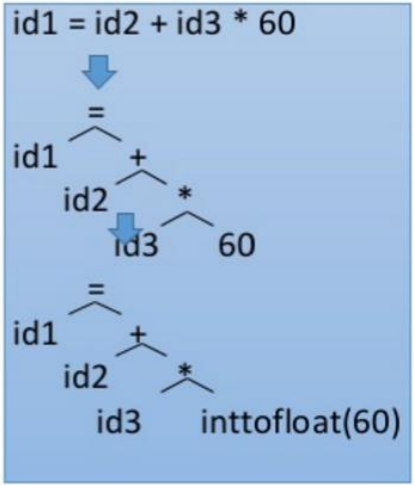


- Se a linguagem traduz todo o texto do programa para só depois executar, então diz-se que o programa foi **compilado**.
- O mecanismo utilizado para a tradução é um **compilador**.
- A versão compilada do programa pode ser executado um número indefinido de vezes sem que seja necessária nova compilação.
  - Compensa o tempo gasto na compilação.

- Reconhece as sequências de símbolos (cadeia de caracteres) que representam uma unidade (sentença).
- Objetivo: separar os símbolos adequadamente.
- Sentenças reconhecidas (**Tokens**):
  - Identificadores, palavras-chaves, variáveis, constantes, operadores, delimitadores.
  - Instruções (`while`, `for`, etc).

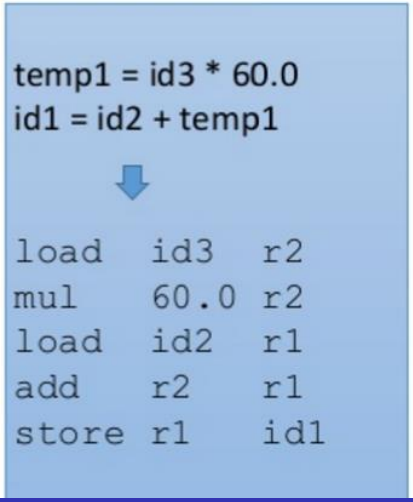


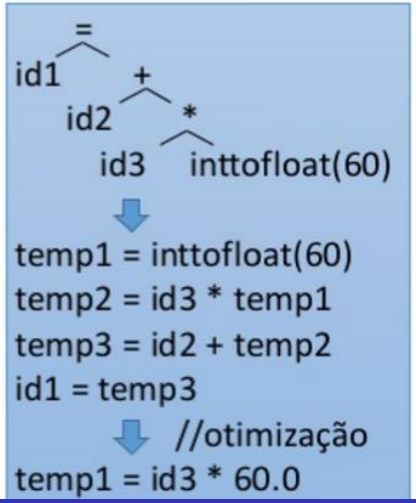
- Recebe os *tokens* e determina se estão dispostos conforme especifica a gramática da linguagem.
- Identifica a estrutura gramatical e o papel de cada componente.
- Resultado: Construção de uma **Árvore Sintática** e uma **Tabela de Símbolos**.



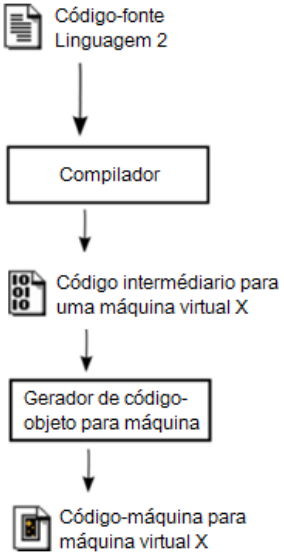
Linguagens Compiladas: Geração de Código

- Processo de construir instruções da linguagem de máquina (em *Assembly*).
- Simulam as instruções reconhecidas na análise sintática.





Linguagens Interpretadas



- Se a linguagem traduz o programa para uma linguagem intermediária (**bytecode**) que só depois será executada, diz-se que o programa foi **interpretado**.
- Utiliza uma **Máquina Virtual** para executar o programa.
- A tradução do bytecode para linguagem de máquina é feita em tempo de execução.
  - A máquina virtual reconhece a arquitetura do hardware e traduz de acordo.

Especificação Formal (BNF)

Especificação Léxica em BNF

- <operador> ::= + | - | * | / | = | != | < | > | <= | >=
- <identificador> ::= <identificador> <letra> | <identificador> <dígito> | <letra>
- <número> ::= <dígito> <número> | <dígito>
- <dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <letra> ::= a | b | c | d | e | f | g | h | i | j | l | m | n | o | p | q | r | s | t | u | v | x | z

Especificação Sintática em BNF

- <programas> ::= { <comandos> }
- <comandos> ::= <comando> <comandos> | <comando>
- <comando> ::= <atribuição> | <condicional> | <loop>
- <atribuição> ::= <identificador> = <expressão>
- <condicional> ::= if <expressão> { <comandos> } | if <expressão> { <comandos> } else { <comandos> }
- <loop> ::= while <expressão> { <comandos> }
- <expressão> ::= <identificador> | <número> | (<expressão>) | <expressão> <operador> <expressão>

FORMAL vs INFORMAL

- As linguagens de programa têm uma **especificação formal** para especificar de forma estruturada seus conjuntos de regras.
- Os métodos **informais**, por sua vez, denotam a inexistência das características associadas ao formalismo:
  - Apresentam frases de significado duvidoso.
  - Geração de manuais imprecisos.
  - Não existem técnicas que auxiliem a implementação.

Especificação Formal

- Precisa e não ambígua.
- Automatizável.
- Permite construir provadores de propriedades de programas.
- Níveis de especificação formal:
  - **Especificação léxica** (Tokens).
  - **Especificação sintática** (Gramática).
  - **Especificação semântica** (Significado).

Especificação Formal: Sintática

- Especifica as regras para determinar se uma dada cadeia de entrada pertence ou não à linguagem definida por uma gramática.
- Produz uma estrutura denominada **Abstract Syntax Tree (AST)**.
- A **Backus-Naur Form (BNF)** é a notação mais utilizada para a descrição sintática.

Especificação Formal: Semântica

- Descreve o significado das estruturas do programa expressos na sua sintaxe.
- A semântica pode ser:
  - **Semântica estática:** Descreve as características de um programa válido.
  - **Semântica dinâmica:** Descreve os resultados da execução do programa.
- **Nota:** Não existe ainda um formalismo aceito globalmente para descrever a semântica da linguagem.

Exemplos de Formalismos Semânticos

- Semântica Operacional Estrutural.
- Máquina de Estado Abstrato.
- Semântica Denotacional.
- Semântica de Ações.

Importante

Mais detalhes serão vistos na disciplina de **Compiladores**.

Por que Programar?

- **É útil:** Pode ser aplicada à Arte, Ciência, Filosofia e Entretenimento.
- **É divertido:** Resolver problemas é como decifrar quebra-cabeças.
- **É uma arte:** Criar algo do zero a partir de lógica pura.

CLP - Conceitos de Linguagens de Programação

Aula 01: Conceitos de Linguagens de Programação

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Março de 2026