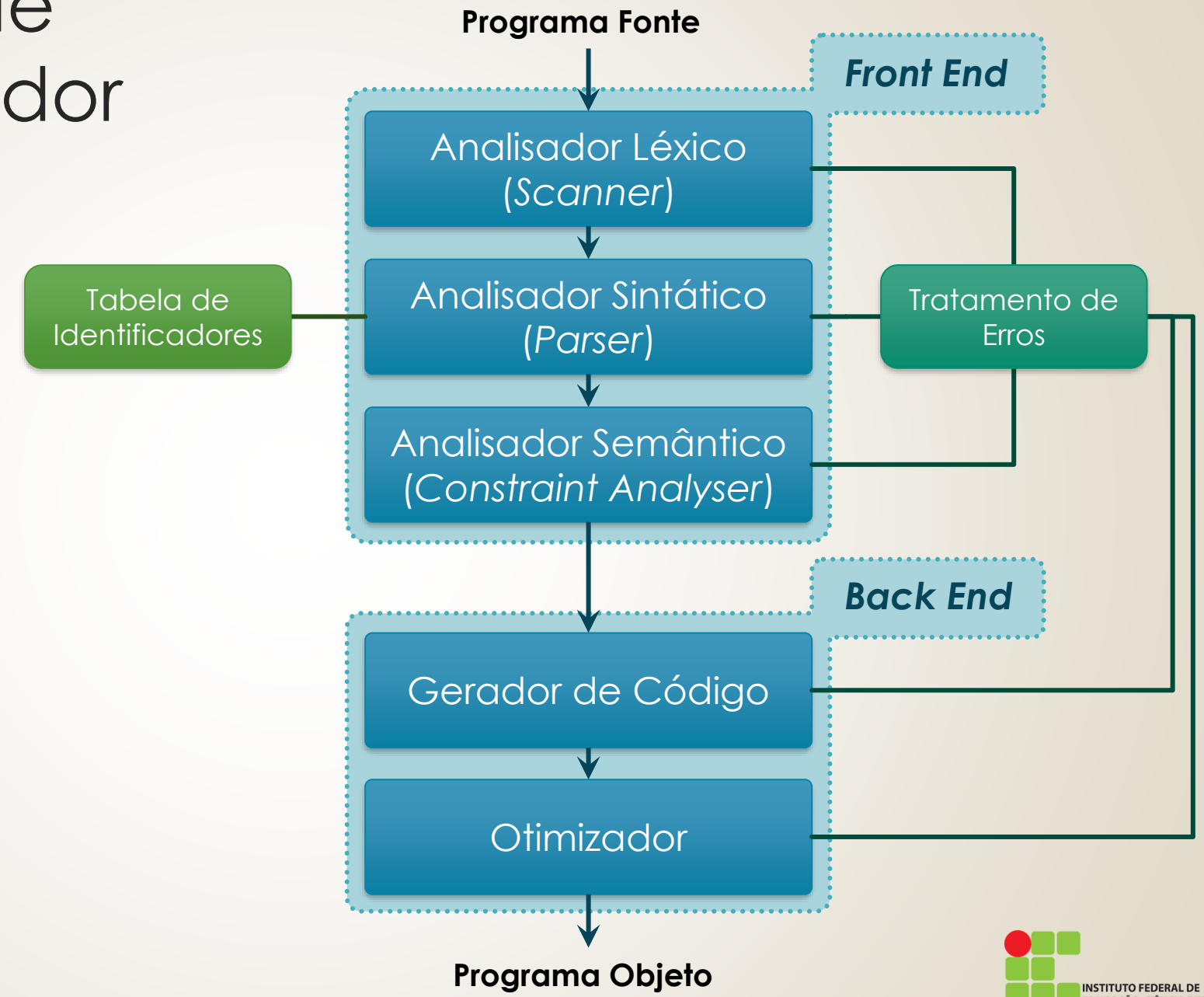
SBVCONC: Construção de Compiladores

Aula 02: A Estrutura de um Compilador

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

A Estrutura de um Compilador



Algumas Observações em Relação à Estrutura dos Compiladores

- Nem toda fase é separada como uma coleção de módulos, por exemplo, a análise sintática e análise semântica podem ser "entrelaçadas";
- Muitos compiladores utilizam algum tipo de representação intermediária durante o processo de compilação:
  - A Árvore Sintática Abstrata, que é a estrutura de dados de alto nível usada na representação da estrutura básica de um programa;
  - Código intermediário de baixo nível similar ao código de máquina, mas independente de *hardware*;
- Algumas otimizações podem ser feitas nas representações intermediárias.

O *Front End* do Compilador

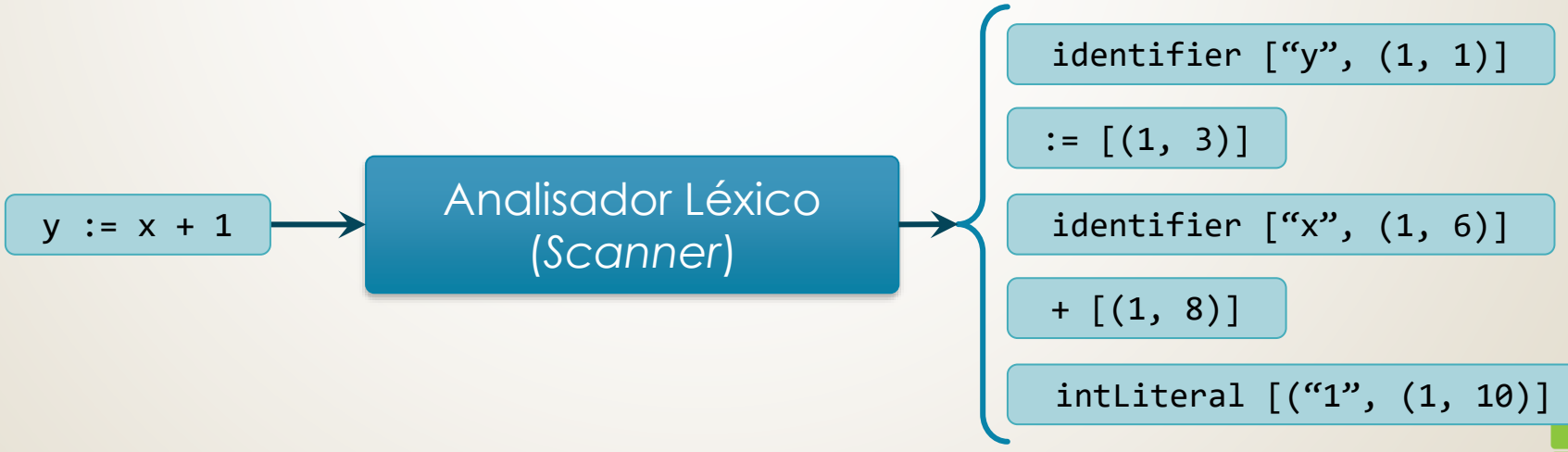
- Análises:
  - Léxica;
  - Sintática;
  - Semântica (restrições);
- Determina se o código fonte é válido;
- Determina o efeito pretendido do programa;
- Altamente dependente da linguagem fonte;
- Relativamente independente da máquina alvo;
- Pode incluir algum tipo de otimizações de alto nível.

O *Back End* do Compilador

- Síntese:
  - Geração de Código;
  - Otimização;
- Gera o código de máquina semanticamente equivalente e razoavelmente eficiente ao programa fonte;
- Relativamente independente da linguagem fonte;
- Altamente dependente da máquina alvo.

Análise Léxica
(*Scanner*)

- Identifica as unidades léxicas básicas da linguagem:
  - Chamadas de *tokens* ou símbolos;
  - Baseado em expressões regulares;
- Remove comentários e espaços em branco excedentes;
- Reporta quaisquer erros.



Análise Sintática
(*Parser*)

- Verifica se as regras gramaticais da linguagem são satisfeitas;
- Baseado em gramáticas livres de contexto, normalmente em notação BNF (Backus-Naur Form).

Análise Semântica
(*Constraint Analyzer*)

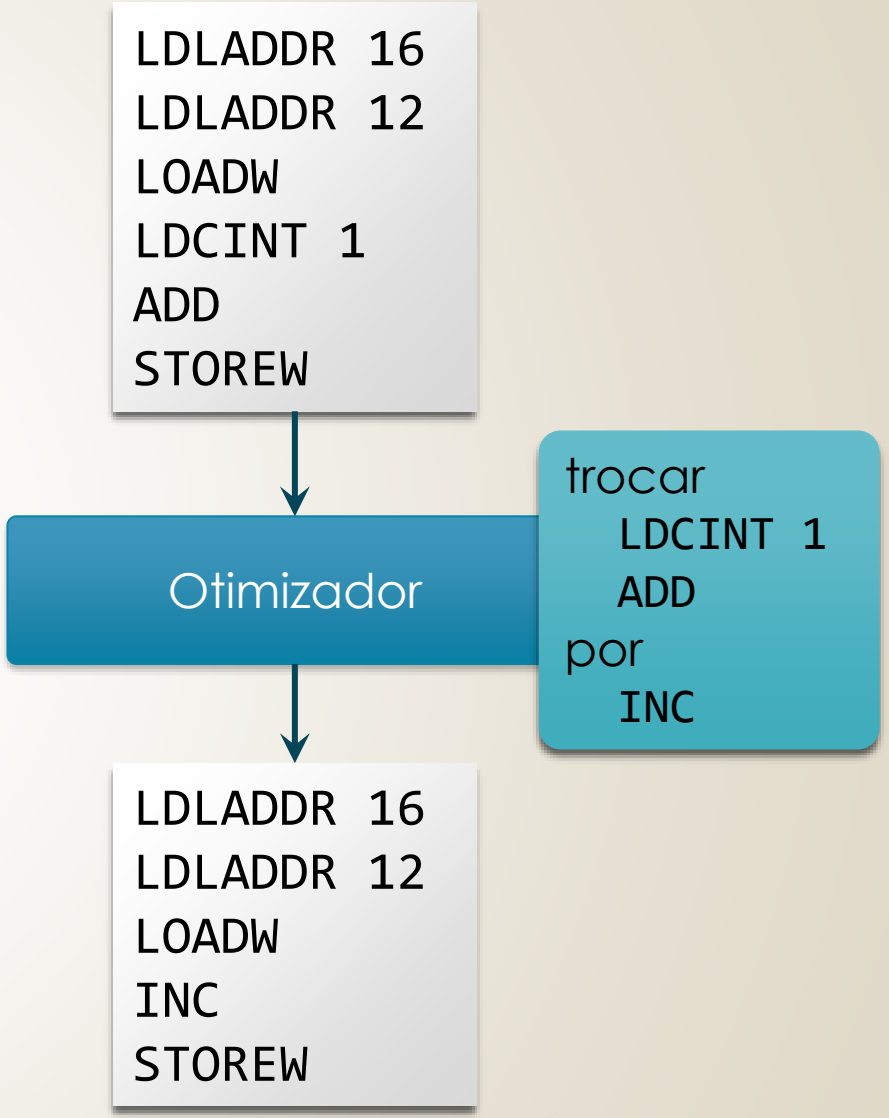
- Trata os requisitos da linguagem que não podem ser expressos em uma gramática livre de contexto, por exemplo, uma variável deve ser declarada apenas uma vez;
- Realiza a análise de tipo e de escopo;
- Normalmente checa a validade com praticamente nenhuma modificação na representação atual;
- A análise semântica é chamada também de análise da semântica estática ou de análise de restrições (*constraint analysis*).

Gerador de Código

- Traduz a representação intermediária em código de máquina ou em linguagem assembly;
- Altamente dependente da máquina.

Otimizador

- Performance do código em relação a tempo e espaço;
- Alocação de registradores;
- Movimentação de computações invariantes para fora de laços;
- Resolução de aritmética em tempo de compilação;
- Otimizações tanto na representação intermediária quanto no código objeto;
- Otimizações locais versus globais;
- Otimização de compiladores.



Tabelas e Mapas

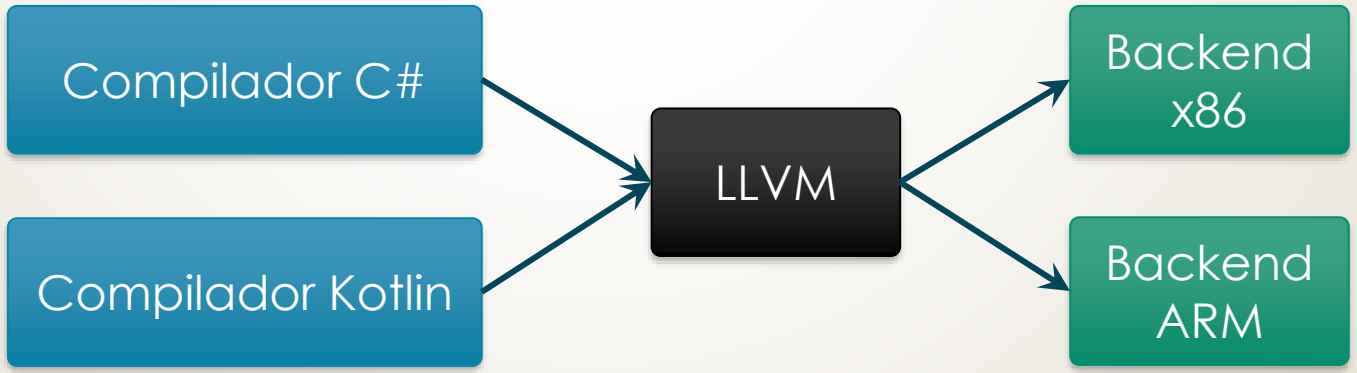
- Conduzem o processo de compilação em parsers baseados em tabelas;
- Armazenam informações dos identificadores (seus atributos):
  - Tabela/Mapa de identificadores;
  - Tabela/Mapa de símbolos;
  - Tabela/Mapa de tipos.

Tratamento de Erros

- Reporta a natureza e a localização de erros (mensagens de erro);
- Na maioria das vezes, um compilador é usado para compilar programas incorretos!
- Recuperação de erros:
  - Turbo Pascal versus abordagem tradicional;
  - Todos os erros reportados após o primeiro são “suspeitos”.

Ferramentas para Construção de Compiladores

- Geradores de parsers (compiladores de compiladores): ANTLR, Coco/R, Flex/Bison, Lex/Yacc, JavaCC etc.
- Motores de tradução dirigida por sintaxe;
- Geradores de código e otimizadores para representações intermediárias de baixo nível: LLVM.



Passadas

- Uma passada consiste em uma travessia completa no programa fonte ou em uma representação intermediária equivalente;
- Na maioria das vezes envolve entrada/saída em disco, mas a representação intermediária pode ser em memória;
- Compiladores de passagem única executam apenas uma travessia no programa fonte;
- Compiladores de múltiplas passadas executam diversas travessias.

Compiladores de Passada Única (*Single-pass*) versus de Múltiplas Passadas (*Multi-pass*)

- Vantagens dos compiladores de múltiplas passadas:
  - Modularidade aumentada;
  - Maior habilidade de realizar análise global (otimização);
  - Normalmente menos memória é necessária em tempo de execução;
  - Ideal para sistemas multiprocessados;
  - Algumas linguagens requerem mais de uma passada, por exemplo se um identificador é definido após ser usado;
- Desvantagem dos compiladores de múltiplas passadas:
  - Podem ser mais lentos, especialmente se estiver envolvida uma quantidade extra de entrada/saída em disco;
  - Normalmente maiores (mais código) e mais complexos;
  - Requerem o projeto de linguagens ou de representações intermediárias.

Passadas no Nosso Projeto de Compilador

- **Passada 1:** Lê e analisa o código fonte, produzindo uma representação intermediária usado árvores sintáticas abstratas;
- **Passada 2:** Executa a análise semântica;
- **Passada 3:** Gera código em linguagem assembly da CVM;
  - Algumas otimizações são feitas pelo assembler (fornecido).
- Todas as passagens intermediárias usam as árvores sintáticas abstratas em memória;
- As operações de entrada/saída somente ocorrem na leitura do código fonte e na escrita do código objeto.

Possíveis Objetivos no Projeto de Compiladores
(Conflitos e Escolhas)

- Confiabilidade (Regra número 1: um compilador tem que ser livre de erros);
- Modularidade/manutenibilidade;
- Programas objeto rápidos e pequenos;
- Tempos de compilação curtos;
- Tamanho do compilador reduzido;
- Boas capacidades de diagnóstico e recuperação de erros;
- Minimização do tempo de desenvolvimento do compilador.

Bibliografia

MOORE JR., J. I. **Introduction to Compiler Design: an Object Oriented Approach Using Java**. 2. ed. [s.l.]:SoftMoore Consulting, 2020. 284 p.

AHO, A. V.; LAM, M. S.; SETHI, R. ULLMAN, J. D. **Compiladores: Princípios, Técnicas e Ferramentas**. 2. ed. São Paulo: Pearson, 2008. 634 p.

COOPER, K. D.; TORCZON, L. **Construindo Compiladores**. 2. ed. Rio de Janeiro: Campus Elsevier, 2014. 656 p.

JOSÉ NETO, J. **Introdução à Compilação**. São Paulo: Elsevier, 2016. 307 p.

SANTOS, P. R.; LANGOLOIS, T. **Compiladores: da teoria à prática**. Rio de Janeiro: LTC, 2018. 341 p.