Apresentação da disciplina

Construção de compiladores I

Objetivos

Objetivos

- Apresentar a importância da construção de compiladores na formação de um cientista da computação.
- Apresentar a ementa, critérios de avaliação e bibliografia da disciplina.

Objetivos

- Apresentar a visão geral de um compilador.

Compiladores

Como executamos programas?

- Três técnicas principais
  - Interpretação
  - Compilação
  - Virtualização.

Interpretação

- Código fonte é representado como uma estrutura de dados e o intepretador executa o programa diretamente.
- Interpretador é responsável pela execução do programa e de sua interação com o S.O.

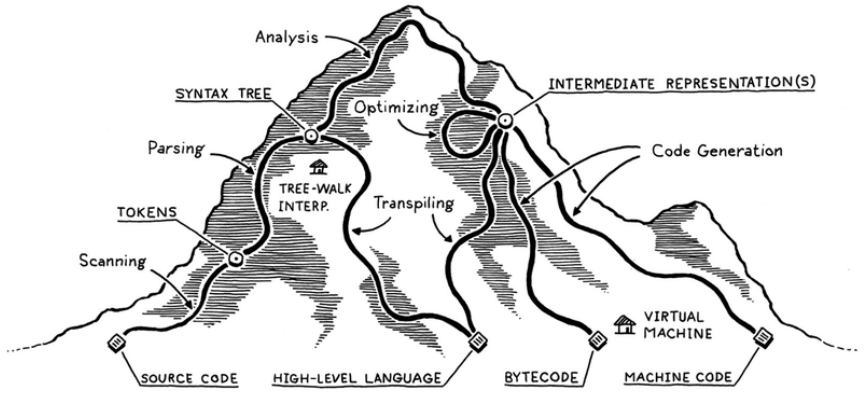
Compilação

- Código fonte é traduzido para uma nova linguagem.
- Código alvo pode ser uma linguagem de alto nível (transpilação).
- Código alvo pode ser uma linguagem de baixo nível (compilação).

Virtualização

- Código fonte é traduzido para uma linguagem de baixo nível, não diretamente entendida pelo hardware.
- Normalmente, chamado de bytecode.
- Intepretadores mais eficientes.

Visão geral (montanha)



Escalando a montanha

- Análise Léxica: Dividir o código em **tokens**.
  - Eliminar comentários e espaços em branco.

Escalando a montanha

- Análise sintática: Organiza a sequência de tokens em sua estrutura gramatical.
  - Produz uma árvore de sintaxe abstrata.

Escalando a montanha

- Análise semântica: Verifica se a árvore produzida pelo analisador sintático atende restrições semânticas.
  - Todos os símbolos foram declarados?
  - Argumentos de funções possuem a aridade e tipo corretos?

Escalando a montanha

- Intepretador: executa o código diretamente a partir da árvore de sintaxe.
  - Não há geração de código
  - Comum em linguagens dinamicamente tipadas, como Python.

Escalando a montanha

- "Transpilador": converte a árvore de sintaxe para outra linguagem de alto nível.
  - Ideal para permitir independência de plataformas.

Escalando a montanha

- Geração de código intermediário: código de baixo nível independente de plataforma.
  - Utilizado para otimizações independentes de hardware.

Escalando a montanha

- Otimização de código.
  - Otimização de consumo de memória, tempo de execução, tamanho de código, etc...

Escalando a montanha

- Geração de código: tradução da representação intermediária para o código de máquina, diretamente executável pelo hardware.

Escalando a montanha

- Virtualização: geração de código para máquinas virtuais como a JVM / EVM.
  - Diferentes plataformas podem executar o mesmo programa utilizando VMs para a plataforma.

Motivação

Porque estudar compiladores?

- Desenvolver um compilador permite consolidar conhecimentos de:
  - Teoria da computação (autômatos e gramáticas)

Porque estudar compiladores?

- Desenvolver um compilador permite consolidar conhecimentos de:
  - Engenharia de software (testes e arquitetura de software)

Porque estudar compiladores?

- Desenvolver um compilador permite consolidar conhecimentos de:
  - Arquitetura de computadores (conhecer detalhes do alvo de compilação)

Porque estudar compiladores?

- Possivelmente, o primeiro artefato de software complexo produzido por estudantes de graduação.

Porque estudar compiladores?

- Compiladores aparecem em toda parte!
  - Navegadores web (JavaScript e WebASM)
  - Monitoramento do Kernel Linux (eBPF)
  - Várias aplicações possuem linguagens para customização.

Porque estudar compiladores?

- Projeto de compiladores envolve problemas difíceis:
  - Executam várias tarefas e devem ser eficientes.
  - Responsáveis por bom uso de uma linguagem.
  - Devem ocultar detalhes de arquitetura e SO de desenvolvedores.

Porque estudar compiladores?

- Provavelmente, uma das áreas mais consolidadas da ciência da computação!

Porque estudar compiladores?

- Vários pesquisadores da área foram agraciados com o Turing Award!
  - John Backus, Barbara Liskov, Niklaus Wirth, Edsger Djikstra, Robin Milner e C.A. Hoare.

Porque estudar compiladores?

- A área de linguagens de programação, apeser de teórica, possui demanda de vagas!
  - Áreas de atuação: ferramentas de análise estática de código e teste automatizado.
  - Verificação formal de aplicações WEB3 (contratos inteligentes).

Porque estudar compiladores?

- Pesquisa em compiladores?
  - Como produzir código mais eficiente? Otimização de código.
  - Como garantir que um compilador é correto? Verificação e teste.
  - Criar novas linguagens e seus compiladores.

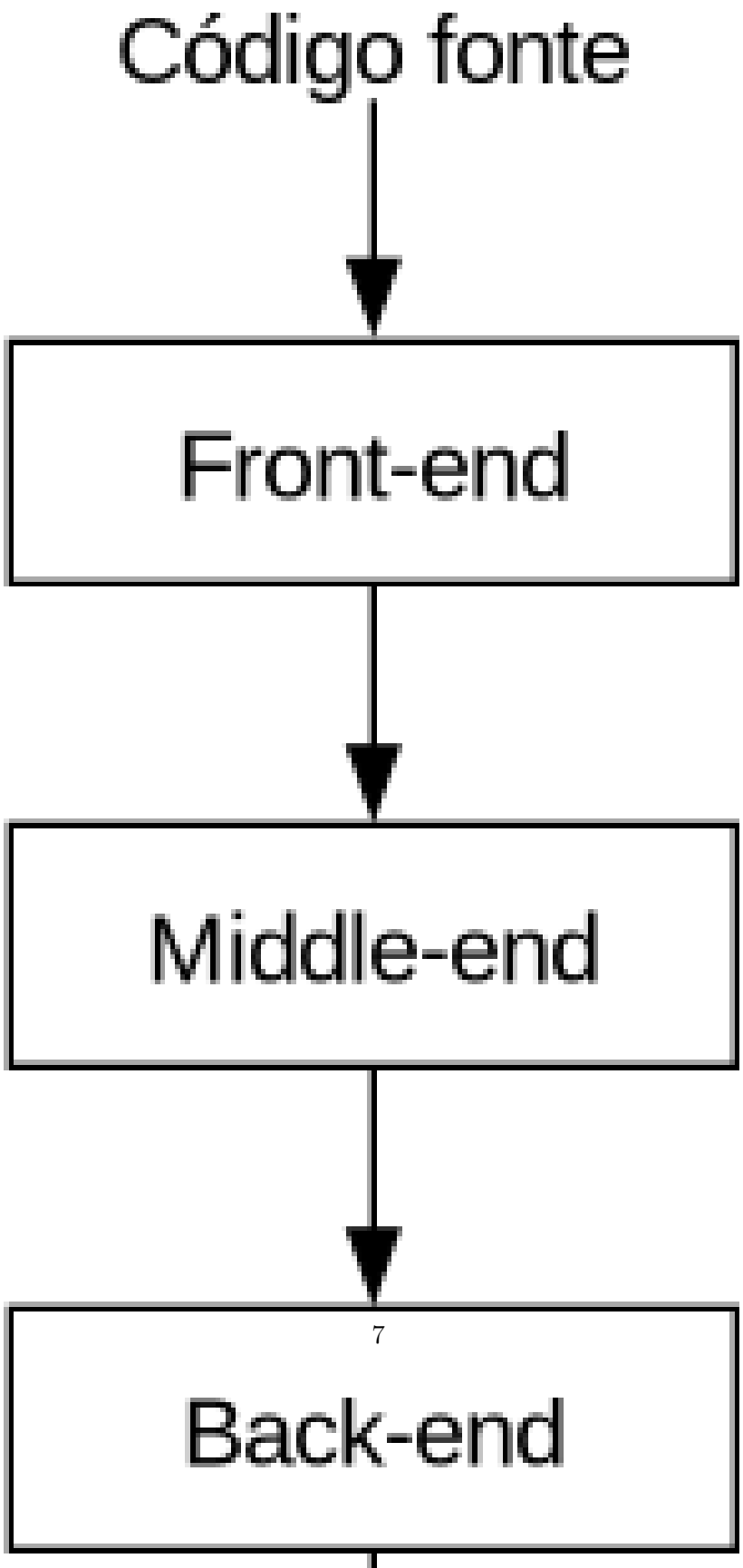
Estrutura de um compilador

Tarefas

- Um compilador deve:
  - Detectar todos os erros e reportá-los
  - Deve preservar a semântica do programa de entrada.
  - Realizar interface do programa com o SO.

Estrutura geral

- Estrutura geral de um compilador


Código fonte
Front-end
Middle-end
Back-end

Front-end de um compilador

- Responsável pela análise do código.
- Produz uma representação intermediária para geração de código.

Middle-end de um compilador

- Responsável por otimizações independentes de arquitetura.
  - Constant folding: substituir 3 + 4 por 7 no código.
  - Dead code elimination: eliminar código que nunca é executado.
  - Loop unrolling: eliminar laços para reduzir custo de desvios.

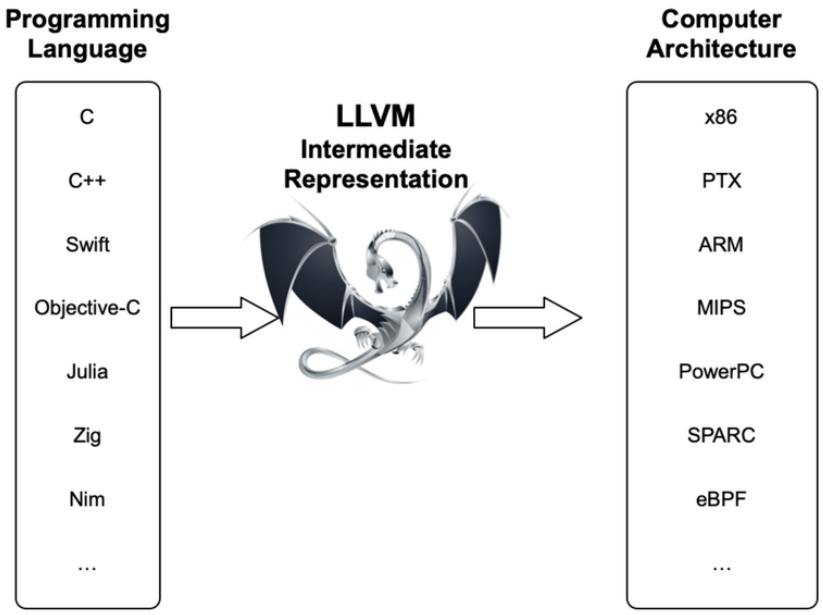
Back-end de um compilador

- Responsável por otimizações dependentes do hardware.
  - Alocação de registradores: como usar registradores da melhor maneira?
  - Escalonamento de instruções: Decidir ordem de instruções para melhor eficiência?
  - Otimizações específicas de arquitetura: usar vectorização, múltiplos núcleos.

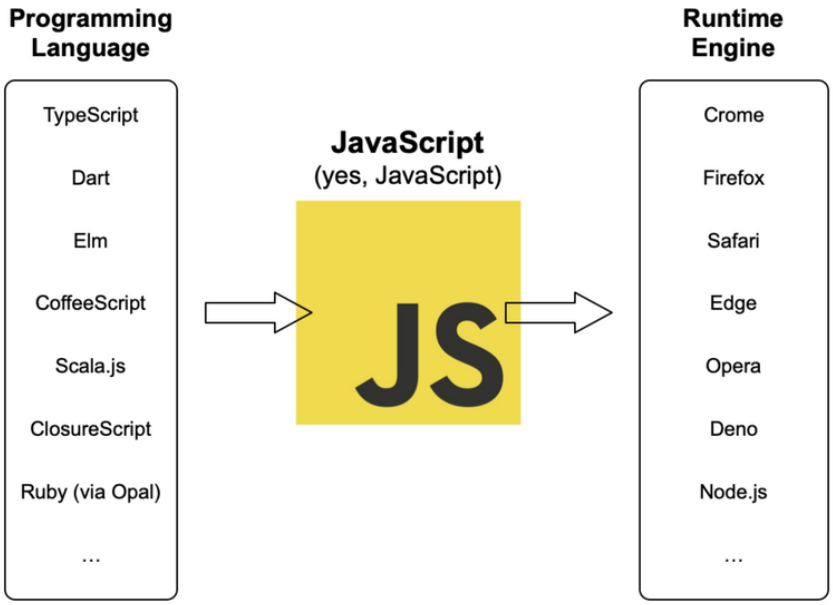
Porque essa arquitetura?

- Um mesmo front-end pode suportar diferentes middle-ends, sem modificações na linguagem fonte.
- Um mesmo middle-end pode suportar diferentes arquiteturas.
- Separação permite que desenvolvimento seja focado em problemas de cada componente.

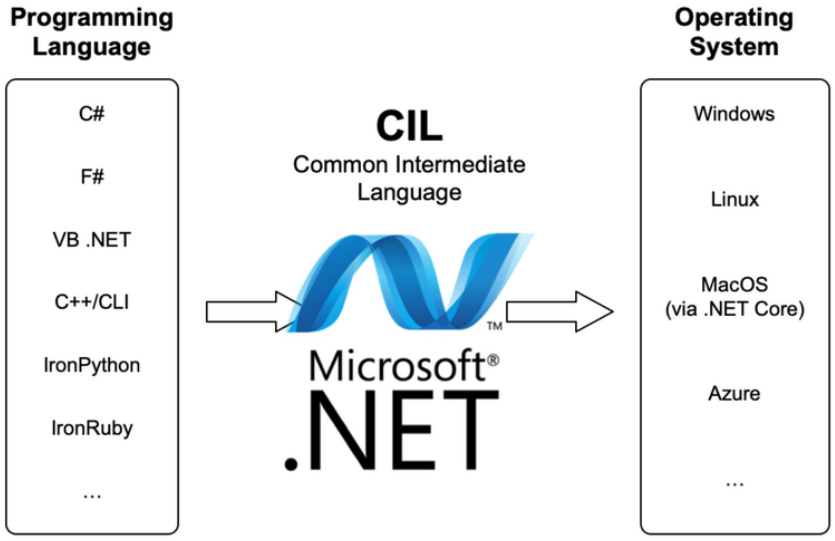
Exemplos



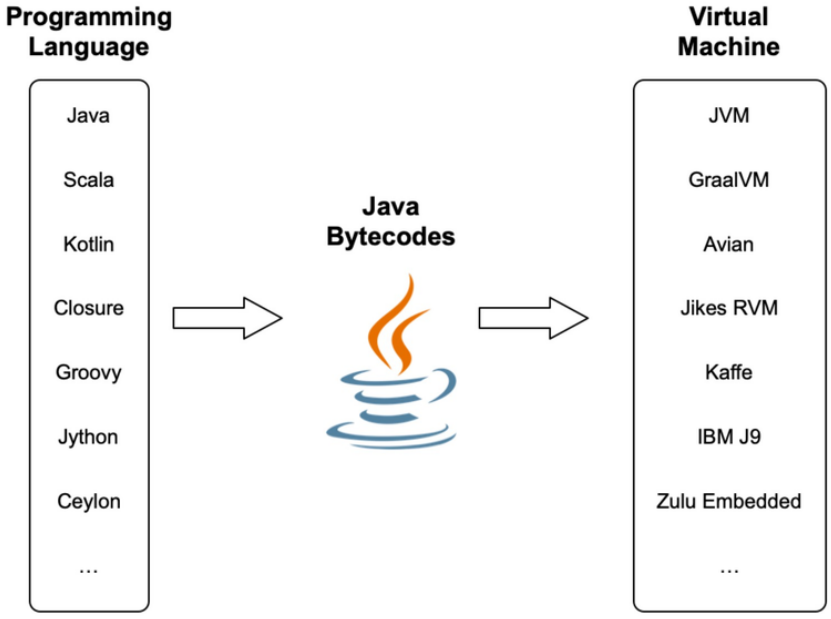
Exemplos



Exemplos



Exemplos



Ementa

Ementa

- Introdução ao processo de compilação e interpretação
- Análise léxica
- Análise sintática
- Análise semântica e geração de código intermediário.

Bibliografia

Bibliografia

- Construindo Compiladores. Cooper, Keith D. ; Torcson, Linda
- Compiladores: Princípios, técnicas e ferramentas. Aho, Alfred; Lam, Monica; Sethi, Ravi; Ullman, Jeffrey.
- Modern compiler implementation in ML. Appel, Andrew.

Materiais de apoio

Materiais de apoio

- Slides e código de exemplo serão disponibilizados no seguinte repositório online.

Critérios de Avaliação

Critérios de Avaliação

- Uma avaliação no valor de 4,0 pontos.
- Trabalhos práticos e exercícios de programação no valor 6,0 pontos.

Critérios de Avaliação

- Avaliação versa sobre o conteúdo teórico da disciplina:
  - Funcionamento de algoritmos

  - Semântica de linguagens de programação
  - Sistemas de tipos

Critérios de Avaliação

- Trabalhos práticos sobre o conteúdo
  - Extensão de protótipos de compiladores apresentados na disciplina.
  - Desenvolvimento de um projeto de ferramenta que utiliza técnicas de compilação.

Critérios de Avaliação

- Entregas de trabalhos
  - Entrega 1: 25/10/2023
  - Entrega 2: 18/11/2023
  - Entrega 3: 29/01/2024

Critérios de Avaliação

- Exercícios de programação
  - Datas de entrega a serem determinadas na plataforma Moodle.

Critérios de Avaliação

- Data avaliação: 05/02/2024

Exame especial

- Mínimo de 75% de frequência e nota inferior a 6,0.
- Exame especial parcial para alunos que perderam uma avaliação.
  - Envolverá tarefas de codificação e atividades teóricas (em papel).
- Detalhes: Resolução CEPE 2880 de 05/2006

Exame especial

- Data do exame especial: 19/02/2024

Software

Software

- Trabalhos e códigos de exemplo serão desenvolvidos utilizando Haskell.
- Utilizaremos diversas bibliotecas da linguagem Haskell.

Outras informações

Informações

- Toda informação da disciplina será disponibilizada na plataforma Moodle.
- Email: rodrigo.ribeiro@ufop.edu.br

Atendimento

- Segunda-feira: 08:00 - 10:00h e 15:30-17:30h.
- Quarta-feira: 08:00 - 10:00h.

Finalizando

- Tenhamos todos um excelente semestre de trabalho!

Conclusão

Conclusão

- Próximas aulas: Análise léxica.

Bibliografia

- Compiladores: Princípios, técnicas e ferramentas. Aho, et.al.
- Crafting interpreters. Nystrom, Robert.