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.

Motivação

Compiladores

- Peça central da ciência da computação.
  - Tarefa: traduzir programas de "alto nível" em código de máquina
  - Tecnologia responsável pelo grande avanço da computação!

Compiladores

- Desenvolver um compilador permite consolidar conhecimentos de:
  - Estruturas de dados e algoritmos (árvores, grafos, etc. . .)

Compiladores

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

Compiladores

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

Compiladores

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

Compiladores

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

Compiladores

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

Compiladores

- Projeto de compiladores envolve problemas difíceis:
  - Executam várias tarefas e devem ser eficientes.

Compiladores

- Projeto de compiladores envolve problemas difíceis:
  - Responsáveis por bom uso de uma linguagem.

Compiladores

- Projeto de compiladores envolve problemas difíceis:
  - Devem ocultar detalhes de arquiteturas de desenvolvedores.

Compiladores

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

Compiladores

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

Compiladores

- Mas porque criar compiladores?
  - Criar novas linguagens para facilitar tarefas de desenvolvimento.
  - Exemplos: Lua, Elixir, Elm, Rust, Scala e outras linguagens

Compiladores

- Conteúdo aplica-se somente a criar novas linguagens?
  - Não! Diversas ferramentas úteis utilizam conceitos de compiladores.

Ementa

Ementa

- Revisão de programação funcional em Haskell
- Introdução ao processo de compilação e interpretação

Ementa

- 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.

Software

- Infraestrutura para desenvolvimento de trabalhos está configurada utilizando o gerenciador de pacotes Nix.
- Gerenciador de pacotes Nix pode ser instalado em Windows, Linux e MacOS.

Software

- Utilizando o Nix, você conseguirá um ambiente consistente para desenvolvimento de trabalhos e exercícios.
  - Versão correta de compilador, bibliotecas e ferramentas auxiliares.

Software

- Recomanda-se **FORTEMENTE** o uso do Nix para garantir o mesmo ambiente de execução para trabalhos.
- Código que não executar no ambiente, não será considerado para correção.

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!

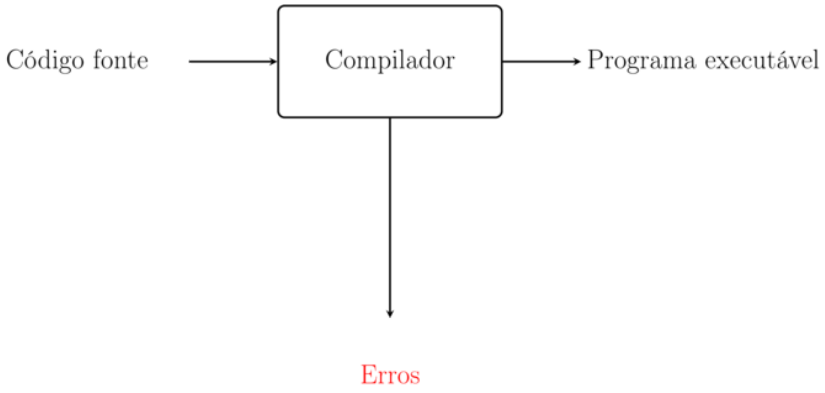
Motivação

Compiladores

- Peça central da ciência da computação.
  - Tarefa: traduzir programas de "alto nível" em código de máquina
  - Tecnologia responsável pelo grande avanço da computação!

Compiladores

- Nosso objetivo, responder a pergunta:
  - Como um compilador funciona?

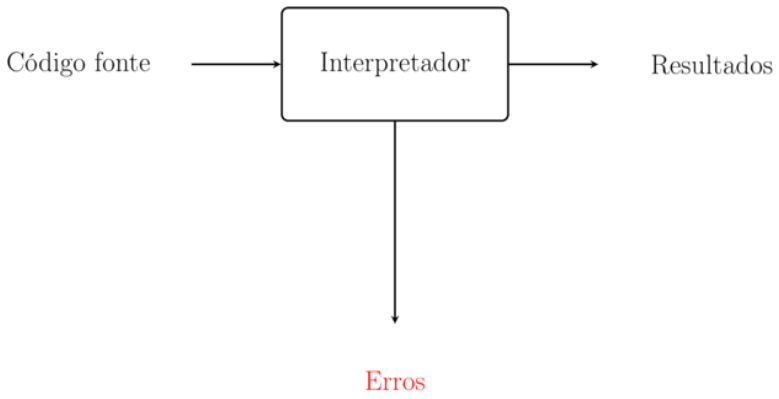


Compiladores

- 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.

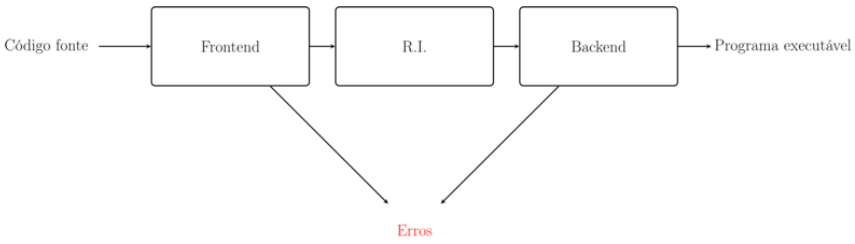
Interpretadores

- Estrutura geral de um interpretador



Compiladores

- Estrutura de alto nível



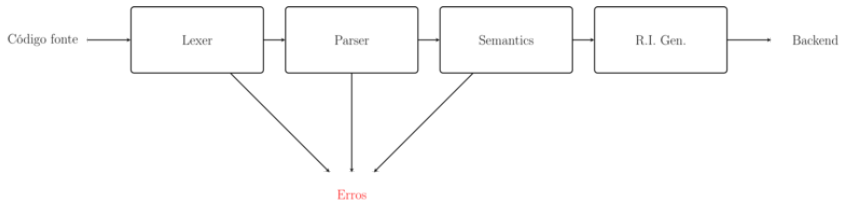
Frontend

Frontend

- Responsável pela análise do código fonte.
  - Deve detectar e reportar todos os erros antes da geração de código.
  - Deve produzir uma representação intermediária do código fonte.

Frontend

- Estrutura do frontend



Estrutura de um frontend

Analisador léxico

- Componente responsável por identificar elementos do "alfabeto" da linguagem
  - Normalmente chamados de "tokens".
- Responsável por eliminar espaços em branco e comentários da entrada.

Analisador léxico

- Capaz de identificar erros simples de digitação de palavras chave.
  - Ex: "wihle" ao invés de "while".
- Resultado: Lista de tokens.
- Formalismo: autômatos finitos e linguagens regulares.

Analisador sintático

- Componente responsável por identificar estrutura gramatical.
- Capaz de identificar uma grande quantidade de erros.
  - Ex. Esquecer um ";" no fim de um comando.
  - Ex. Não usar parêntesis balanceados.

Analisador sintático

- Resultado: Árvore de sintaxe abstrata (AST).
- Formalismo: Gramáticas livres de contexto e autômatos de pilha.

Analisador semântico

- Responsável por validar regras semânticas da linguagem
  - Todo uso deve possuir declaração correspondente.
  - Regras de tipo.
- Resultado: AST com anotações de tipos.

Gerador de IR

- Responsável por converter a AST em uma representação um pouco mais próxima do código final.
- Em algumas situações, a própria AST é uma possível IR.

Gerador de IR

- Neste curso, vamos usar a LLVM.
  - Framework para construção de backends usando a forma SSA.
- Também veremos como gerar código para uma máquina de pilha.
  - Simplificação da JVM / EVM.

Conclusão

Conclusão

- Próximas aulas: revisão de Haskell
  - Compilador de um subconjunto de markdown.