Compiladores (CC3001) — Aula 1: Apresentação

Mário Florido

DCC/FCUP

2024

[dcc] FC FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Esta aula

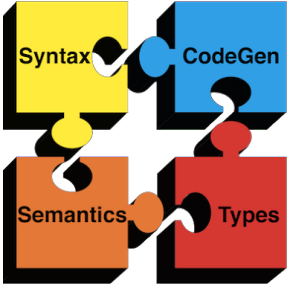
Objetivos e funcionamento

Compiladores e interpretadores

Extras

Conteúdos e objetivos

1. Princípios de conceção e construção de compiladores
   - análise léxical
   - análise sintática
   - árvore sintática abstrata
   - geradores de analisadores lexicais e sintáticos
   - tabelas de símbolos
   - invocação de funções e registos de ativação
   - geração de código intermédio
   - alocação de registos e geração de código máquina



2. Trabalho laboratorial: implementação de um compilador para uma linguagem imperativa simples

Funcionamento

Aulas teóricas

- Presenciais no anfiteatro
- Exposição de conceitos e exemplos

Aulas laboratoriais

- Presenciais nos laboratórios
- Resolução de exercícios
- Acompanhamento do trabalho
- *Moodle* e *GitHub Classroom* para exercícios e trabalho

Ferramentas

- Recomendado sistema operativo GNU/Linux
- Windows: use o WSL (*Windows Subsystem Linux*)
- *haskell-platform*: GHC, alex, happy, bibliotecas padrão
- Pode também usar Java ou C
- O seu editor de texto/IDE predileto
- *Git* (linha de comando)

Avaliação

Trabalho prático: 15% + 15% (2 fases)

Exame final: 70%

- Os exercícios de aulas são individuais
- Trabalho prático:
  - **componente obrigatória**
  - realizado em grupo de 2 estudantes
  - entregas e apresentações: 1ª fase em novembro, 2ª fase em dezembro
- Classificação mínima no exame final: 40% (8 valores em 20)

Bibliografia recomendada

1. *Basics of Compiler Design*, Torben Aegidus Mogensen: `http://hjemmesider.diku.dk/~torbenm/Basics/basics_lulu2.pdf`
2. *Modern Compiler construction in ML*, Andrew W. Appel, Cambridge University Press (existem também versões em C e Java.)
3. *Compilers: Principles, Techniques, and Tools*, Alfred Aho, Jeffrey Ullman, Ravi Sethi, Monica Lam

Esta aula

- Tradutor de programas numa linguagem de programação para outra
- Usualmente: traduz uma linguagem de *alto nível* numa de *baixo nível*
- Principal técnica para implementação de linguagens

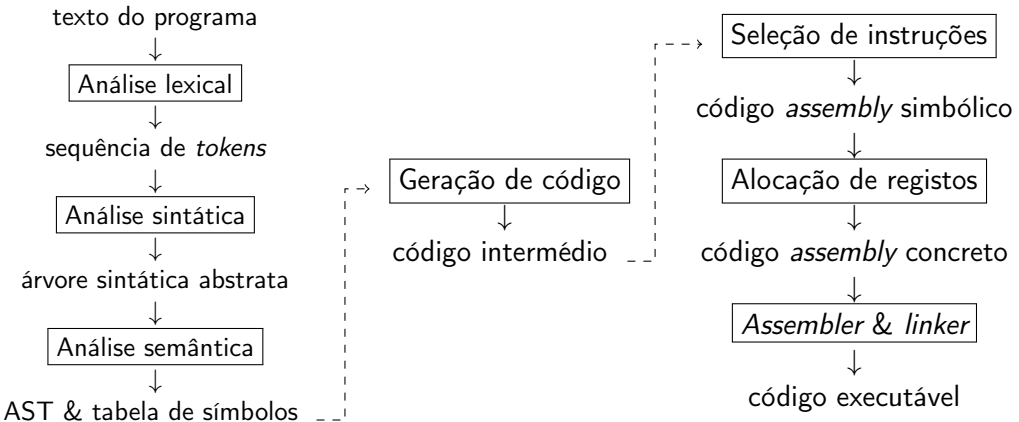
Alguns compiladores

	linguagem fonte	**linguagem destino**
GCC	C/C++/...	código máquina[1]
Clang	C/C++/...	código máquina
Fpc	Pascal	código máquina
GHC	Haskell	código máquina
Rustc	Rust	código máquina
Javac	Java	código JVM[2]
Scalac	Scala	código JVM
elm	Elm	JavaScript

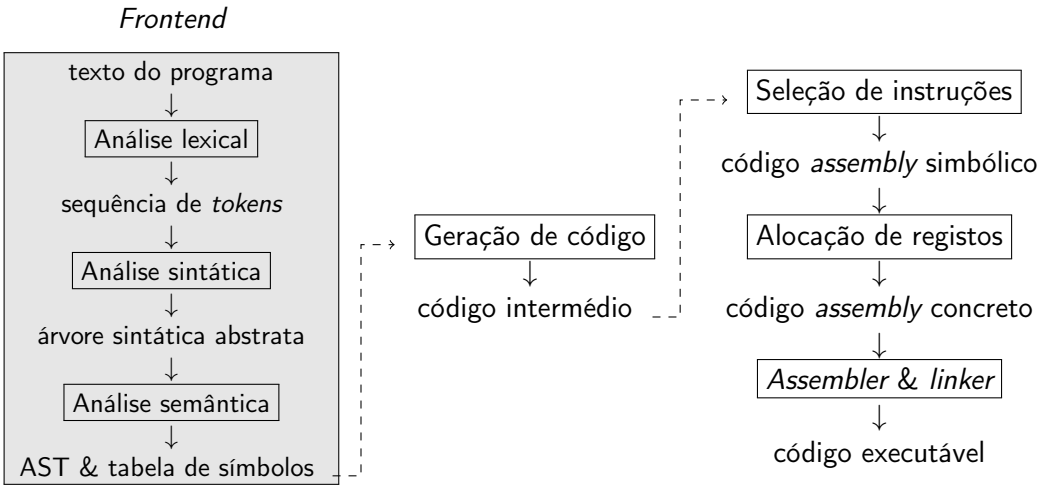
[1] E.g. X86, ARM, ...

[2] *Java Virtual Machine*

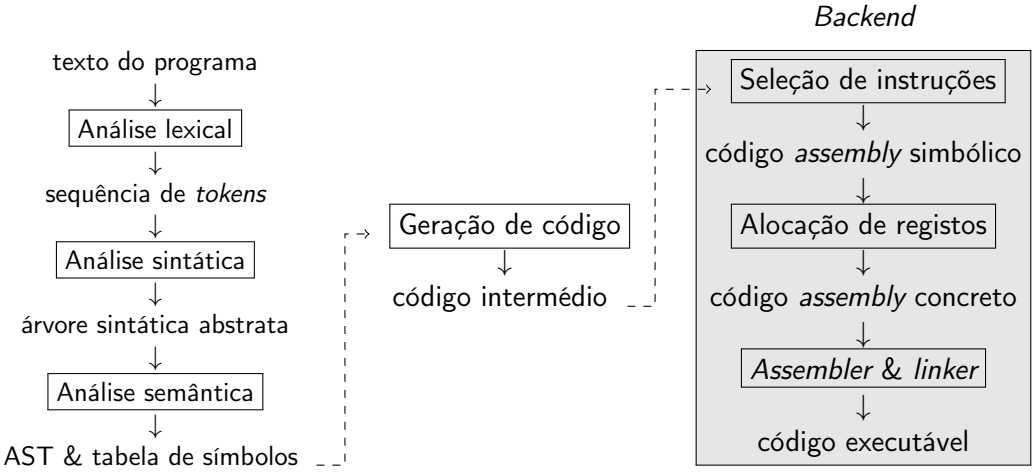
Fases de um compilador



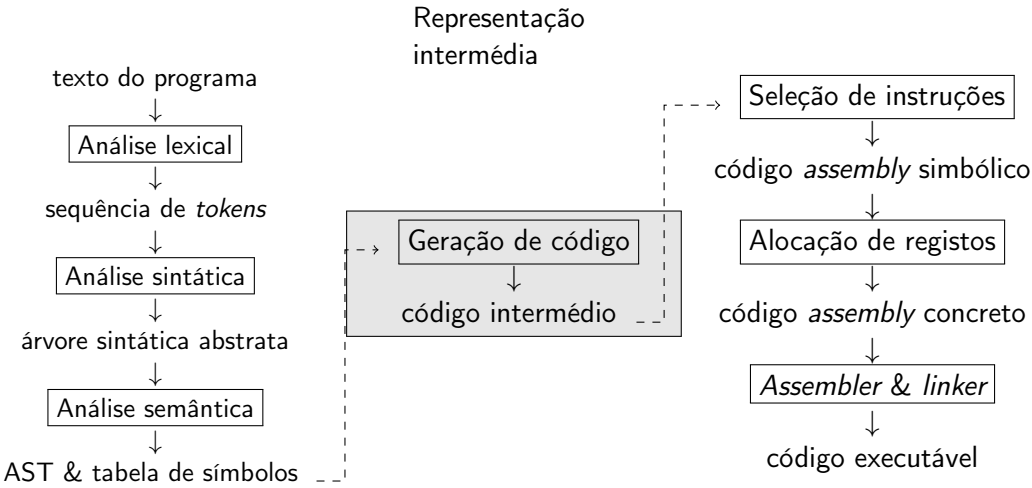
Fases de um compilador



Fases de um compilador



Fases de um compilador



Decomposição em fases

- As fases de *frontend* lidam com linguagem fonte
- As fases de *backend* lidam com a geração de código máquina
- A fase intermédia é independente da linguagem fonte e do código máquina
- Esta decomposição torna o compilador mais simples
- Permite re-utilização de componentes
  - o *backend* do GCC é usado para compilar C, C++ ou Objective-C
  - o *backend* LLVM é usado pelo Clang, Swift, Rustc e (opcionalmente) GHC
- Extras: optimizações em código intermédio e/ou no *backend*

Em vez de um compilador, podemos implementar um *interpretador*:

- efetua a análise lexical e sintática da linguagem fonte (tal como um compilador);
- executa diretamente o programa usando a árvore sintática
- em alternativa: pode gerar e executar um código intermédio

Compiladores vs. interpretadores (cont.)

Vantagens da interpretação:

- Mais simples do que a compilação
- Mais fácil suportar diferentes arquiteturas de computadores
- Suporta desenvolvimento interativo (*read-eval-print-loop*)

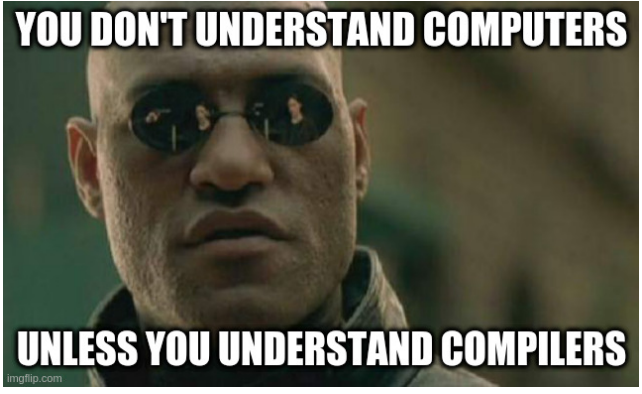
Vantagens da compilação:

- Código compilado pode ser bastante mais eficiente
- Executáveis podem ser distribuidos separamente do compilador

Algumas implementações combinam compiladores e interpretadores (e.g. GHC, OCaml).

A maioria dos programadores não vai escrever o seu próprio compilador.

Porquê então estudar compiladores?

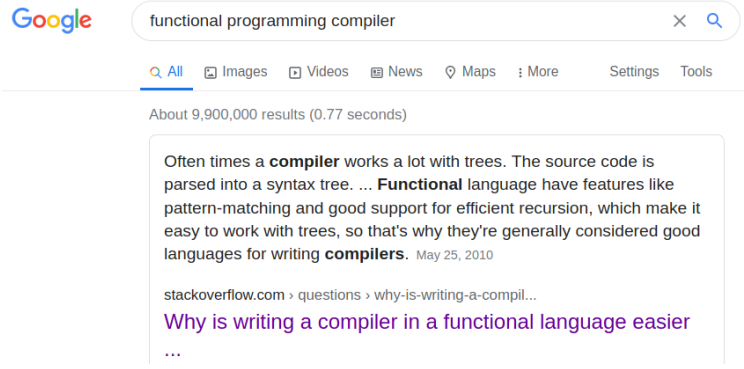

YOU DON'T UNDERSTAND COMPUTERS
UNLESS YOU UNDERSTAND COMPILERS
imgflip.com

Porquê estudar compiladores? (cont.)

- Compreender a ligação entre linguagens de alto e baixo nível
- Técnicas de *parsing* e interpretação são úteis em muitas aplicações
- É possível que tenha de implementar um compilador ou interpretador para uma *linguagem específica de domínio*
- Ficará mais capaz de escrever código correto
- Ficará mais capaz de escrever código eficiente

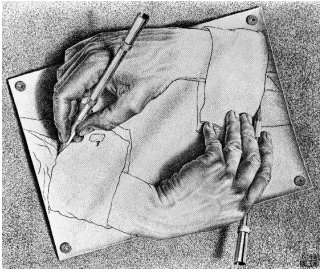
Como implementar compiladores?

- São programas complexos, logo é preferível usar uma linguagem de alto-nível
- Efetuam transformações entre formatos
  (código fonte $\longrightarrow$ AST $\longrightarrow$ código intermédio $\longrightarrow$ código máquina)
- Linguagens funcionais fortemente tipadas são boas escolhas

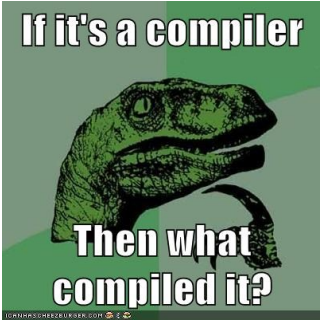


- No entanto: compiladores de linguagens de proposito genérico são tipicamente escrito na própria linguagem ("self-hosting")
- Exemplos:

GCC	é escrito em C
Javac	é escrito em Java
GHC	é escrito em Haskell
Ocamlc	é escrito em OCaml
Rustc	é escrito em Rust



- Prova da maturidade da linguagem e ferramentas (“eat your own dog food”)
- Facilita comunicação entre *developers* (uma só linguagem)



Problema: necessitamos de um compilador para compilar o compilador...

Exemplo histórico: a linguagem C.

- O primeiro compilador de C (1973) foi baseado no compilador de B (uma linguagem anterior)
- O primeiro compilador de B foi implementado numa outra linguagem de alto nível (TMG) do sistema *Multics*
- O compilador foi re-escrito em B por estágios até ser capaz de compilar a si próprio...

Fonte: *The Development of the C language*, Dennis Ritchie;
`https://www.bell-labs.com/usr/dmr/www/chist.html`.

Segundo exemplo: o compilador de Rust.

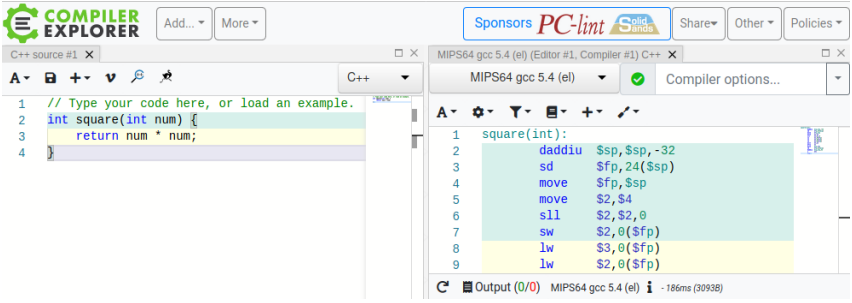
2006-2010: primeiros compilador escrito em OCaml

2010: inicio de re-escrita do compilador em Rust (ainda compilado pelo compilador em OCaml)

Desde 2011: o compilador é desenvolvido em Rust e *self-hosted*

Fonte: *Wikipedia*
`https://en.wikipedia.org/wiki/Rust_(programming_language)`.

O que faz um compilador real?



- Compiler explorer: `https://godbolt.org/`
- Várias linguagens, compiladores e arquiteturas
- Permite também comparar efeitos de otimizações
- Vídeo tutorial: `https://youtu.be/4_HL3PH4wDg`