Como funcionam compiladores

Desde o código-fonte ao binário

Introdução

- Uma breve explicação do tema

Objetivos

1. Perceber o que é um compilador
2. Distinguir os tipos de compiladores
3. Entender as fase de compilação

O que é um compilador ?

- Converte código-fonte para binário
- Também pode converter para objetos de código

Tipos de compiladores

1. Native compiler
2. Bytecode compiler
3. Just-in-time compiler
4. Source-to-source compiler
5. Hardware compiler

Native compiler

- Converte código-fonte em código binário
- Código nativo específico de um arquitetura de processador (x86, ARM, MIPS, ...)

Exemplos

1. GNU Compiler Collection (GCC)
2. Microsoft Visual C++ (MSVC)
3. Clang + LLVM

Bytecode compiler

- Converte código-fonte em instruções em bytecode
- O código em bytecode corre numa maquina virtual que interpreta estas instruções

Exemplos

1. Roslyn C#
2. Javac
3. CPython interpreter

Just-in-time compiler

- Converte, em runtime, código bytecode em código máquina
- Compila parte ou todo o código bytecode
- Usados para otimizar rotinas muito usadas frequentemente

Exemplos

1. PyPy Python Compiler
2. Java Hotspot
3. LuaJIT

Source-to-source compiler

- Converte código-fonte em código-fonte de outra linguagem de programação.
- Também conhecido como transpilador

Exemplos

1. Typescript compiler (Typescript -> Javascript)

Hardware compiler

- Converte código-fonte em description objects, linguagem usada para descrever o hardware

- Usado para programar FPGAs ou construir ASICs.

Exemplos

1. GHDL
2. Compilers das empresas produtoras de hardware

Fases de compilação

1. *Lexer*
2. *Parser*
3. Semantica
4. *Intermediate code generation*
5. *Optimizer*
6. *Target code generation*

Lexer / Scanner

- Separa o código-fonte em tokens

Parser

- Transforma os tokens separados em uma árvore de *syntax* abstrata (AST) com uma gramática definida

Análise Semântica

- Fase mais complexa do compilador
- Converte a AST incompleta numa AST completa
- Infere os tipos e verifica se as regras estão todas em conformidade

Intermediate Code Generation

- Passa a AST completa em uma representação genérica e intermédia de instruções de máquina

Optimizer

- Fase opcional do compilador
- Converte o código intermédio em código intermédio mais optimizado
- Reduz o número de instruções e variáveis em memoria

Target Code Generation

- Converte o código intermédio em código de máquina na arquitetura desejada.

Conclusão

- Importante saber como funcionam os compiladores
- Perceber as diferenças entre os vários tipos e as suas fases de compilação

Referências

- Aho, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986. ISBN 0-201-10088-6
- Parsons, Thomas W., Introduction to Compiler Construction, Computer Science Press, 1992. ISBN 0-7167-8261-8