CSSE 404: Compilers Introduction

MICHAEL WOLLOWSKI

Many, but not all of the materials in this presentation are from the book Compilers, by Dave and Dave

Why Study Compilers?

Brings together much of computer science:

- Algorithms
- Theory
- Architecture

Large, complex project

Central to our work as programmers

Extraordinarily cool when it works

Genealogy of Programming Languages

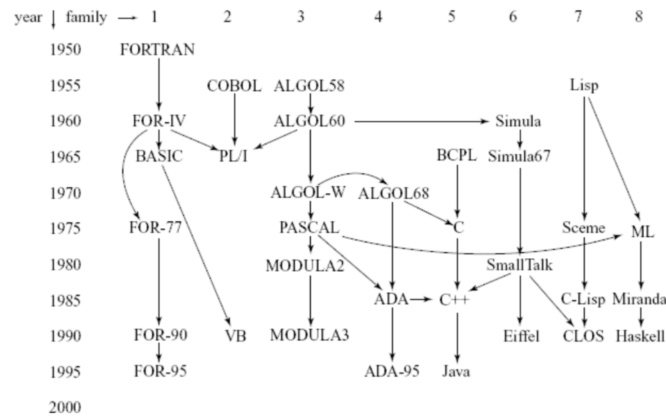

Fig. 1.1 Genealogy of programming languages

Image source: Compilers by Dave and Dave

High Level Languages (HLL)

An HLL looks more like a natural language than a machine or assembly language

Structured programming and modularity are emphasized

Examples: C, C++, Java, Ada

Very High Level Languages (VHLL)

A VHLL is developed for specific application areas or computational needs.

Examples: Prolog, MATLAB, R

Why High Level Languages?

Readability
- ideally self-documenting

Portability
- same source code can be used on different families of machines

Productivity
- Rule of thumb: programmer can deliver 50 lines/day of tested, debugged code

Debugging ease
- Logical meaning of constructs

Why High Level Languages?

Modularity

- Aids in team work and code reuse

Optimization

- Converting tail calls into iterative constructs

Generality

- Write code for many applications

Error detection

- Think generics in Java: detecting as many errors as possible at compile time

What is a Compiler?

A compiler is a *translator*

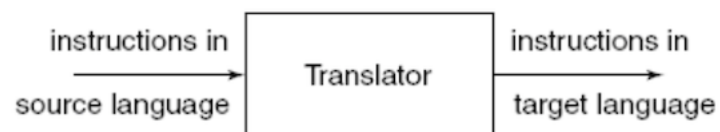

Fig. 1.2 What does a translator do?

Image source: Compilers by Dave and Dave

Fundamental Principles

Preserve meaning of input program

Improve on the input program

If you know how your compiler works, you may be able to take advantage of that.

For example, write a tail-recursive procedure which will be converted to an iteration.

Phases of a Compiler

Pre-processing
- A macro processor which provides for file inclusion, conditional compilation control etc.

Lexical analysis
- Also called the “scanner”
- Identifies and tags the words in the source file
- Example: number, identifier

Syntax analysis
- Also called the “parser”
- Typically produces a parse tree
- Enables the compiler to verify that the source file is a valid program and to check for errors

Phases of a Compiler

Semantic analysis
- Also called the "mapper"
- Processed the parse tree to generate intermediate code

Code generation
- Processed intermediate code to to produce code in some target language
- Target language can be a VM, assembly or machine code

Error checking
- This is spread throughout the compiler

Optimization
- This too is spread throughout the compiler

Basic Structure

Typically composed of stages

Stages grouped into major phases
- Front end
- Optimizer
- Back end

Components of a Compiler

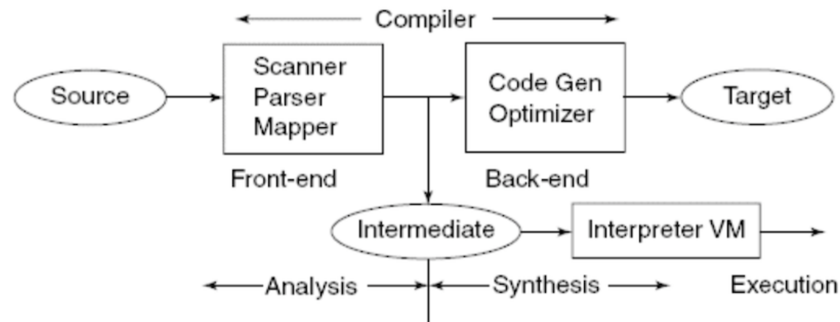

Fig. 1.11 Front- and back-end of a compiler

Image source: Compilers by Dave and Dave

Types of Compilers

One-pass
- Compiler completes all of its processing in one fell swoop
- Benefit: Simple compiler

Multi-pass
- Compiler processes various intermediate representations several times.
- Benefit: More powerful optimization

Nano-pass
- Each phase of the compiler does one thing and one thing only
- Reference: https://www.cs.indiana.edu/~dyb/pubs/nano-jfp.pdf