Class 01: An Introduction to The Course

Held: Friday, 26 August 2011

Summary: We begin the course with an introduction to compilation, and overview of the compilation process and a consideration of the structure of the course.

Related Pages:
- EBoard.

Notes:
- Assignment for Monday: Introductory Survey.
- Reading for Monday: Skim [The Programming Language Pascal: 1973 revised edition]
- Any student plays etc. this weekend?
- The Web site is still under development. (Yeah, I know that’s a big surprise.)

Overview:
- What is a Compiler?
- Why Study Compilers?
- How to Study Compilers.
- Administrative Issues.
- The Project.
- Stages of Compilation.

Background
- I believe in beginning my courses with a short dialog concerning the topic of study. In particular, I think we should ask ourselves:
  - What are compilers?
  - Why do we study compilers? (Why do we have a course in compilers? Why do most CS departments include compilers in the undergraduate curriculum? Why have you chosen to take this class?)
  - How should we study compilers?
- You can find some of my notes on these questions below. However, this class will develop its own answers to these and related questions.
What are compilers?

- Compilers are programs that translate from one language (typically a high-level language, like Java, C, or Pascal) to another (typically a low-level language, like assembler, machine code, JVM, or C).
- To do this translation, they step through a number of phases, which we will discuss later in today’s class.

Why study compilers?

- The joy of creating a significant computing artifact.
- CS is a field that involves both practice (e.g., programming) and theory (e.g., CSC341). Compilers are a wonderful “platform” for studying both practice and theory. In fact, compilation motivated many early theoretical problems.
- Many large programs include their own language. If you are ever called upon to help design such a language, you can draw upon these skills.
  - Increasingly, large programs include one of a few standard “scripting languages”
- Understanding how compilers work may help you program better (although I strongly discourage you from “coding for the compiler”, particularly since newer compilation techniques may actually do worse with programs designed for earlier compilers).
- Compilation involves many important subfields of computer science, including
  - Formal language theory
  - Programming language design
  - Software design
  - Architecture

How should we study compilers?

- Do problems to ensure that we understand the underlying theories.
- Read the code for someone else’s compiler.
- Build one.

Course Organization and Policies

- There are many of accompanying documents which you can find through the handouts page. The introductory survey encourages you to explore more.
- In addition to group programming, you will also do a number of written homeworks (often, simultaneous with the programming work). You should do those homeworks as individuals, although you may discuss ideas and answers with each other.
- I’d like to remind you that academic honesty is an important issue at Grinnell.
  - Please don’t cheat.
  - Please make sure to cite any code you reuse or modify.
  - Please make sure to cite any help you receive.
Project Goal

- We will be working to write a program that translates from Pascal to some appropriate low-level language (probably a simulated simple assembly language (SAL), possibly real x86).
- We should be able to get to the SAL level, but we might also get a little bit further, depending on interests and background.
- I know that CS362 has seemed overwhelming to students in the past. I’ll be working hard this semester to provide additional resources (including code).

Compilation Steps

- A compiler really does a series of steps in compilation (in fact, it is only toward the end that it resembles what we normally mean by “compilation”).
- We think of two big phases
  - The front end translates from a high-level language to an intermediate language.
  - The back end translates from the intermediate language to object code or executable.