CS412/413

Introduction to Compilers and Translators
Spring '00

Lecture 1: Overview

Outline

• About this course
• Introduction to compilers
  – What are compilers?
  – Why should we learn about them?
  – Anatomy of a compiler
• Introduction to lexical analysis
  – Text stream to tokens

Course Information

• Lectures
  – MWF 10:10 - 11:00AM in Hollister 110
• Faculty: Andrew Myers
• Teaching assistants: Nate Nystrom, Alexey Kliger, Andrew Lin
• Course e-mail: cs412@cs.cornell.edu
• Course web page: http://courses.cs.cornell.edu/cs412/2000sp

CS 413 is required!

Textbooks

• Required text
  – Modern Compiler Implementation in Java. Andrew Appel.
• Optional texts
  – Compilers – Principles, Techniques and Tools. Aho, Sethi and Ullman (The Dragon Book)
  – Advanced Compiler Design and Implementation. Steve Muchnick.
• Java reference
• On reserve in Engineering Library

Work

• Homeworks: 4, 20% total
  – 5/5/5/5
• Programming Assignments: 6, 50%
  – 5/7/8/10/10/10
• Exams: 2 prelims, 30%
  – 15/15
  – No final exam
Homeworks

- Three assignments in first half of course; one homework in second half
- Not done in groups—you may discuss however

Projects

- Six programming assignments
- Groups of 3-4 students
  - same grade for all
- Group information due Wednesday
  - we will respect consistent preferences
- Java will be implementation language

All Assignments

- Due at beginning of class
- Late homeworks, programming assignments increasingly penalized
- Project files must be available at noon on the same day

Why take this course?

- CS412 is an elective course
- Expect to learn:
  - practical applications of theory
  - parsing
  - deeper understanding of code
  - manipulation of complex data structures
  - how high-level languages are implemented in machine language
  - a little programming language semantics
  - Intel x86 architecture, Java
  - how to be a better programmer (esp. in groups)

What are Compilers?

- Translators from one representation of a program to another
- Typically: high-level source code to machine language (object code)
- Not always
  - Java compiler: Java to interpretable bytecodes
  - Java JIT: bytecode to executable image

Source Code

- Source code: optimized for human readability
  - expressive: matches human notions of grammar
  - redundant to help avoid programming errors
  - often non-deterministic

```java
int expr(int n)
{
    int d;
    d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```
**Machine code**

- Optimized for hardware
  - Redundancy, ambiguity reduced
  - Information about intent lost
- Assembly code ≈ machine code

**How to translate?**

- Source code and machine code mismatch
- Some languages farther from machine code than others (“higher-level”)
- Goal:
  - high level of abstraction
  - best performance for concrete computation
  - reasonable translation efficiency (<< O(n^3))
  - maintainable code

**Example (Output assembly code)**

Unoptimized Code

Optimized Code

**Correctness**

- Programming languages describe computation precisely
- Therefore: translation can be precisely described (a compiler can be correct)
- Correctness is very important!
  - non-trivial: programming languages are expressive
  - implications for development cost, security
  - this course: techniques for building correct compilers

**How to translate effectively?**

- Series of program representations
- Intermediate representations optimized for program manipulations of various kinds (checking, optimization)
- More machine-specific, less language-specific as translation proceeds

**Idea: Translate in Steps**

- Series of program representations
- Intermediate representations optimized for program manipulations of various kinds (checking, optimization)
- More machine-specific, less language-specific as translation proceeds
What is Lexical Analysis?

- Converts character stream to token stream
  \[
  \text{if (x1 * x2 < 1.0)} \{ \\
  \quad y = x1; \\
  \}
  \]

Token stream

- Gets rid of whitespace, comments
- \(<\text{Token type, attribute}>\)
- <Id, “x”> <Float, 1.0e0>
- Token location preserved for debugging, error messages (line number)

- Issues:
  - how to specify tokens?
  - how to implement tokenizer