

# CS 415: Lecture 1

- Introduction

## Some Important Facts

- Lectures: M&W 7:40-9:00pm (sheesh)
- Recitations: W 9:00-10:00pm (double sheesh)
- Instructor: Thu Nguyen
  - Office hours: TBA, CoRE 326
- TA: TBA
- Course web:  
<http://www.cs.rutgers.edu/~tdnguyen/cs415/>
- Newsgroup: ru.nb.dcs.classes.415
- Hint: check the web page and read the newsgroup  
REGULARLY

## Course Overview

- Topics in the design of programming language translators, including scanning, parsing, semantic analysis, error recovery, code generation, and code optimization
- Official Prerequisites
  - CS 211 (Computer Architecture)
  - CS 314 (Principles of Programming Languages)
- Practical Prerequisites
  - Willingness to participate in class & office hours
  - Unix programming
  - C (or willingness/ability to learn quickly)

## Topics

- Lexical Analysis (scanning)
- Syntax Analysis (parsing)
- Context-sensitive Analysis
- Intermediate Representations
- Code Generation
- Code Optimization

## Books

- Required: Aho, Sethi, and Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986 & 1988.
- Recommended: Levine, Mason, and Brown. *lex & yacc*. O'Reilly, 1990 & 1992.

## What To Expect

- The good stuff
  - ~5-6 paper&pencil assignments
  - Project
    - ┆ We'll build a compiler, probably in 4 phases
    - ┆ You will work together in groups of 2
      - Choose a partner by the end of next week
    - ┆ To be done in C on Unix
- The unavoidable stuff
  - Midterm and final

## Grading

- What I don't think is so important ... what you always want to know ...
- Basis for grades
  - 15% homework
  - 20% mid-term exam
  - 25% final exam
  - 40% project

## Lectures

- This class is as good as YOU make it
  - Do the assigned reading before class ...
    - ... yeah, who am I kidding, right?
    - I know it's hard to do but it makes class much more interesting
  - You need to participate: ask questions, generate discussion, etc.
    - I use overhead transparencies
    - If you don't ask questions almost certainly, I will go too fast
- Lecture slides will be posted on the class web
  - You still need to take notes
- The book is a reference. However not all lecture material will come from the book.

## Recitations

- Recitations will be devoted to homeworks and the project
- We'll work on problems together during this time
- We'll also discuss the project
- If you are not prepared to talk about homeworks and/or the project,

## Collaboration

- Project: although you will be working in teams, you are responsible for understanding *every* phase of the project
- HWs: I encourage you to discuss problems together. This is how you learn. However use the Gilligan's Island principle!
- Academic Honesty
  - I don't look for cheating - I assume that it almost never happen
  - The corollary is, I will not tolerate any form of academic dishonesty

## Translators

- Two general classes of translators, compilers and interpreter
- What is a compiler? Why build a compiler?
- What is an interpreter? Why build an interpreter?
- What about preprocessors? What are those? Why do we need them?

## Compiler

- What is a compiler?
  - A program that translates a program in one language into a program in another language
  - A compiler typically (but not always) lowers the level of abstraction of the translated program
- Why build compilers?
  - How would you like to write MS Office in machine language? (Note that an assembler is really a compiler with a weird name.)

## Interpreters

- What is an interpreter?
  - ┆ A program that maps a program in a particular language to its results
- Why build interpreters?
  - ┆ Architecture independence executable (e.g., Java)
  - ┆ Avoid compilation during software development process, which can be time consuming
  - ┆ Easier to build
  - ┆ Suppose you have a brand new architecture and no cross-compiler (what is a cross-compiler, BTW?), what would you do?
- Compilers and interpreters share many components

## Why Study and Build Compilers?

- Compilers and interpreters are much more common than you might think
  - ┆ What are some examples?
- Compilers embody a wide range of theoretical techniques
- Compilers are essential to programming today's computers
- Compiler construction teaches programming and software engineering skills

## Role of Compiler

- **Abstract away low-level machine details**
  - ┆ Instruction selection, addressing modes, register & cache management, instruction-level parallelism
- **Provide higher-level constructs**
  - ┆ Increase programmer productivity
  - ┆ Better maintenance
  - ┆ Portable

## Compiler Construction

- **Compilers are large, complex programs**
- **By working on compilers, you'll learn**
  - ┆ Interesting algorithms
  - ┆ As well as programming skills such as the use of
    - ┆ Programming tools: compilers, debuggers
    - ┆ Program-generation tools: lex, yacc
    - ┆ Software libraries: sets, collections
  - ┆ Hopefully, you will also enhance your software engineering skills

## What Do You Want in a Compiler?

## What Do You Want in a Compiler?

- Correct output
- Output runs fast
- Output runs in predictable amount of time
- Compiler runs fast
- Compile time proportional to program size
- Support for separate compilation
- Good diagnostics for errors
- Works well with debugger
- Cross language calls

## My Biases

- Compiler is important to what I do, which is systems and security
- However, I ain't a compiler person
- The corollary is ... question what I tell you!

## Some Advice

- Do the reading before lectures
- Start early on the project
- Ask lots of questions
- Start early on the project
- Come to office hours
- Start early on the project
- Start early on the project

## Next Time



- Please read *ASU Chapters 1 and 2*
- Check out the class web site
- Read the newsgroup