Overview of the Course
Welcome to CISC 471 / 672 — Compiler Construction

*Topics in the design of programming language translators, including parsing, semantic analysis, error recovery, code generation, and optimization*

- Instructor: Dr. John Cavazos (cavazos@cis.udel.edu)
- Office Hours: Mon 3-4PM / Wed 3-4PM or by appointment
- Office Hours Location: Saxbys Coffee (Amstel Ave)
  - Project handouts, lecture slides, online documentation, ...
  - I will not have handouts in class; get them from the web
Difference between CISC471 and CISC672

Two main differences:

1. CISC471 have less challenging projects

1. CISC471 have less challenging midterm and final

However, this will likely be the hardest class you take!
Basis for Grading

- Exams
  - Midterm 20%
  - Final 30%

- Projects
  - Cool Test Programs 4%
  - Scanner 5%
  - Parser 8%
  - Semantic Analyzer 14%
  - Code Generation 15%

This only adds up to 96%. Where is the other 4%?

Class participation!

Notice: Any student with a disability requiring accommodations in this class is encouraged to contact me after class or during office hours, and to contact UDel’s Coordinator for Disabled Student Services.
Basis for Grading

- Exams
  - Midterm
  - Final
- Projects
  - Parser & Scanner
  - Semantic Analyzer
  - Code Generation

- Closed-notes, closed-book
- First two projects (Test codes and Scanner) are individual projects
- Last three projects to be done in teams
- High ratio of thought to programming
- Will build a compiler for a language called COOL (Java)
Rough Syllabus

- Overview § 1
- Scanning § 2
- Parsing § 3
- Context Sensitive Analysis § 4
- Inner Workings of Compiled Code § 6, 7
- Introduction to Optimization § 8
- Instruction Selection § 11
- Instruction Scheduling § 12
- Register Allocation § 13
- More Optimization (*time permitting*)
Class-taking technique for Course

- I will use projected material extensively
  → I will moderate my speed, *you* sometimes need to say “STOP”

- You should read the book
  → Not all material will be covered in class
  → Book complements the lectures

- You are responsible for material from class
  → The tests will cover both lecture and reading
  → I will probably hint at good test questions in class

- This is not a programming course
  → Projects are graded on functionality, documentation, and lab reports more than style (*results matter*)

- It will take me time to learn your names (*please remind me*)
Compilers

- What is a compiler?
Compilers

- What is a compiler?
  - A program that translates a program in one language into a program in another language
  - The compiler should improve the program, in some way
- What is an interpreter?
Compilers

• What is a **compiler**?
  → A program that translates a program in one language into a program in another language
  → The compiler should improve the program, *in some way*

• What is an **interpreter**?
  → A program that reads a program and produces the results of executing that program
Compilers

• What is a compiler?
  → A program that translates a program in one language into a program in another language
  → The compiler should improve the program, in some way

• What is an interpreter?
  → A program that reads a program and produces the results of executing that program

• C is typically compiled, Scheme is typically interpreted

• Java is compiled to bytecodes (code for the Java VM)
  → which can then interpreted
  → Or a hybrid strategy is used
    ▪ Just-in-time compilation
Taking a Broader View

• Compiler Technology
  → Offline
    ▪ Typically C, C++, Fortran
  → Online
    ▪ Typically Java, C##

→ Goals: improved performance and language usability
  ▪ Making it practical to use the full power of the language

→ Trade-off: preprocessing time versus execution time (or space)

→ Rule: performance of both compiler and application must be acceptable to the end user
Why Study Compilation?

- **Compilers are important system software components**
  - They are intimately interconnected with architecture, systems, programming methodology, and language design
- **Compilers include many applications of theory to practice**
  - Scanning, parsing, static analysis, instruction selection
- **Many practical applications have embedded languages**
  - Commands, macros, formatting tags ...
- **Many applications have input formats that look like languages,**
  - Matlab, Mathematica, Databases (e.g., Oracle)
- **Writing a compiler exposes practical algorithmic & engineering issues**
  - Approximating hard problems; efficiency & scalability
**Intrinsic interest**

- Compiler construction involves ideas from many different parts of computer science

| Artificial intelligence | Greedy algorithms  
|-------------------------|------------------  
|                         | Heuristic search techniques |
| **Algorithms**          | Graph algorithms,  
|                         | Dynamic programming |
| **Theory**              | DFAs & PDAs, pattern matching  
|                         | Fixed-point algorithms |
| **Systems**             | Allocation & naming,  
|                         | Synchronization, locality |
| **Architecture**        | Pipeline & hierarchy management  
|                         | Instruction set use |
Intrinsic merit

- Compiler construction poses challenging and interesting problems:
  - Compilers must do a lot but also run fast
  - Compilers have responsibility for run-time performance
  - Compilers are responsible for making it acceptable to use the full power of the programming language
  - Computer architects perpetually create new challenges for the compiler by building more complex machines
  - Compilers must hide that complexity from the programmer
  - Success requires mastery of complex interactions of compiler phases
Aren’t compilers a solved problem?

“Optimization for scalar machines is a problem that was solved ten years ago.”

David Kuck, Fall 1990
Aren’t compilers a solved problem?

“Optimization for scalar machines is a problem that was solved ten years ago.”

David Kuck, Fall 1990

- Architectures keep changing
- Languages keep changing
- Applications keep changing
- When to compile keeps changing
About the instructor

• My own research
  → Applying machine learning to solve hard systems problems
  → Compiling for advanced microprocessor systems
  → Interplay between static and dynamic compilation
  → Optimization for embedded systems (space, power, speed)
  → Interprocedural analysis and optimization
  → Nitty-gritty things that happen in compiler back ends
  → Distributing compiled code in a heterogeneous environment
  → Rethinking the fundamental structure of optimizing compilers

• Thus, my interests lie in
  → Building “Intelligent” Compilers
  → Quality of generated code (smaller, more efficient, faster)
  → Interplay between compiler and architecture
  → Static analysis to discern program behavior
  → Run-time performance analysis
Next class

- The view from 35,000 feet
  - How a compiler works
  - What I think is important
  - What is hard and what is easy