

Overview of the Course



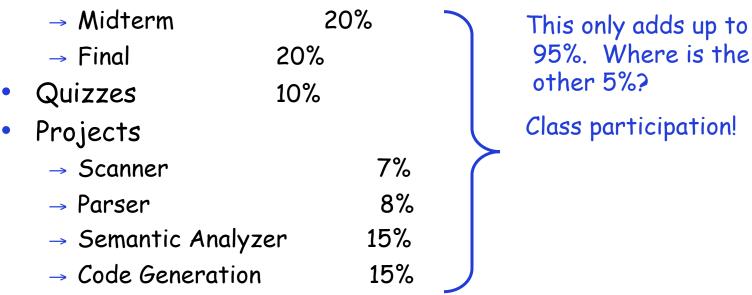
Welcome to CISC 672 — Advanced Compiler Construction

<u>Topics</u> 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: Tues/Thurs 11 AM to 12 PM, Smith Hall 412
- Text: Engineering a Compiler by Keith Cooper and Linda Torzcan
- Web Site: <u>http://www.cis.udel.edu/~cavazos/CISC672</u>
 - → Handouts, homework, slides, ...
 - \rightarrow I will not have handouts in class; get them from the web

Basis for Grading

• Exams



<u>Notice:</u> 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 	 Closed-notes, closed-book Old exam on web site as an example
• Quizzes	 Reinforce concepts Number of quizzes <i>t.b.d.</i>
 Projects → Parser (& scanner) → Semantic Analyzer → Code Generation 	 Parser lab might be a team lab High ratio of thought to programming Will build a compiler for a language called COOL (in C++ or 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)	

More Optimization (*time permitting*)



Class-taking technique for CISC 672

ELAWARE T7 4 3 *

- I will use projected material extensively
 - → I will moderate my speed, you sometimes need to say "STOP"
- You should read the book
 - \rightarrow Not all material will be covered in class
 - \rightarrow Book complements the lectures
- You are responsible for material from class
 - \rightarrow The tests will cover both lecture and reading
 - \rightarrow I will probably hint at good test questions in class
- CISC 672 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)

• What is a compiler?





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



- What is a compiler?
 - → A program that translates an *executable* program in one language into an *executable* program in another language
 - → The compiler should improve the program, in some way
- What is an interpreter?
 - → A program that reads an *executable* program and produces the results of executing that program



- What is a compiler?
 - → A program that translates an *executable* program in one language into an *executable* program in another language
 - → The compiler should improve the program, in some way
- What is an interpreter?
 - → A program that reads an *executable* 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)
 - \rightarrow which can then interpreted
 - \rightarrow 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##
 - \rightarrow 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
 - \rightarrow 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
- Writing a compiler exposes practical algorithmic & engineering issues
 - → Approximating hard problems; efficiency & scalability



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



- Compiler construction poses challenging and interesting problems:
 - \rightarrow 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
 - \rightarrow Success requires mastery of complex interactions

Aren't compilers a solved problem?



"Optimization for scalar machines is a problem that was solved ten years ago." David Kuck, Fall 1990



"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 SPEC CPU?
- When to compile keeps changing



- My own research
 - → Applying machine learning to solve hard systems problems
 - → Compiling for advanced microprocessor systems
 - \rightarrow Interplay between static and dynamic compilation
 - → Optimization for embedded systems (*space, power, speed*)

 - → Nitty-gritty things that happen in compiler back ends
 - → Distributing compiled code in a heterogeneous environment
 - \rightarrow Rethinking the fundamental structure of optimizing compilers
- Thus, my interests lie in
 - → Building "Intelligent" Compilers
 - → Quality of generated code(smaller, more efficient, faster)
 - \rightarrow Interplay between compiler and architecture
 - → Static analysis to discern program behavior
 - \rightarrow Run-time performance analysis

Next class

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

