

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Overview of the Class

Copyright 2010, Pedro C. Diniz, all rights reserved.
Students enrolled in the Compilers class at the University of Southern California (USC) have explicit permission to make copies of these materials for their personal use.

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Critical Facts

Welcome to Compilers — *Compiler Design and Implementation*

Topics in the design of programming language translators, including parsing, run-time storage management, error recovery, code generation, and optimization

- Instructor: Dr. Pedro C. Diniz (pedro@isi.edu)
- Office Hours: Wednesdays, 2.30 - 3.30 PM, SAL 234
- Textbook: **“The Dragon Book” not required, but very helpful.**
- Web Site: <http://www.isi.edu/~pedro/Teaching/CSCI565-2010>
 - Projects, homework, slides, sample solved exercises ...
 - I will not have handouts in class; get them from the web in electronic format

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Basics of Grading

- Exams
 - Midterm (in class) 20%
 - Final (in class) 30%
- Homeworks (5) 25%
- Programming Projects
 - Developing Your Own Compiler/Language
 - Semantic Analysis 10%
 - Translation and Code Generation 15%
 - Website has a lot of info and code!
 - How to Get Started.
 - Links to Lex, Yacc.
 - Basic Data Structures (linked-list, arrays, graphs,...)

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Tentative Syllabus

- Introduction & Overview
- Lexical Analysis: Scanning
- Syntactic Analysis: Parsing
- Syntax-Directed Translation & Parse Tree
- Intermediate Code Generation
- Control-Flow Analysis
- Semantic Analysis and Error Checking
- Run-Time Environment & Storage Organization
- Data-Flow Analysis
- Code Generation
- Instruction Scheduling
- Register Allocation
- More Optimizations (*time permitting*)

Lecture 1 → HW 1
 Lecture 2 → HW 2
 Lecture 3 → HW 3
 Lecture 4 → HW 3
 Lecture 5 → Project 1
 Lecture 6 → HW 4
 Lecture 7 → HW 4
 Lecture 8 → Project 2
 Lecture 9 → HW 5
 Lecture 10 → HW 5

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Class-taking Technique for Compilers

- I will use projected material extensively
 - I will moderate my speed, *you* sometimes need to say “STOP”
- You should read the notes before coming to class
 - 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
- Compilers is a programming course
 - Projects are graded on functionality, documentation, and lab reports more than style (*results matter*)

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

On-line Material for the Class

- Class Web Site
 - <http://www.isi.edu/~pedro/Teaching/CSCI565-2010>

Compilers - Compiler Design
Spring 2009

Contents

- Home
- Lecture Notes
- Lecture 1: Introduction, Parsing and Control Flow Graphs (Feb. 2009)
- Lecture 2: Lexical Analysis (Feb. 2009)
- Lecture 3: Syntactic Analysis: Parsing (Feb. 2009)
- Lecture 4: Syntax-Directed Translation (Feb. 2009)
- Lecture 5: Intermediate Code Generation (Feb. 2009)
- Lecture 6: Control-Flow Analysis (Feb. 2009)
- Lecture 7: Semantic Analysis and Error Checking (Feb. 2009)
- Lecture 8: Run-Time Environment & Storage Organization (Feb. 2009)
- Lecture 9: Data-Flow Analysis (Feb. 2009)
- Lecture 10: Code Generation (Feb. 2009)
- Lecture 11: Instruction Scheduling (Feb. 2009)
- Lecture 12: Register Allocation (Feb. 2009)
- Lecture 13: More Optimizations (Feb. 2009)

- Class Forum on DEN

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Schedule of the Class

USC Spring 2010	Monday	Tuesday	Wednesday	Thursday	Friday
8:00 - 8:30					
8:30 - 9:00					
9:00 - 9:30					
9:30 - 10:00					
10:00 - 10:30					
10:30 - 11:00					
11:00 - 11:30					
11:30 - 12:00					
12:00 - 12:30					
12:30 - 13:00					
13:00 - 13:30					
13:30 - 14:00					
14:00 - 14:30					
14:30 - 15:00			Office Hours SAL 214		
15:00 - 15:30			Lectures		
15:30 - 16:00			OHIE 122		
16:00 - 16:30			Office Hours SAL 214		
16:30 - 17:00					
17:00 - 17:30					
17:30 - 18:00					
18:00 - 18:30					
18:30 - 19:00					
19:00 - 19:30					
19:30 - 20:00					
20:00 - 20:30					

Legend: ■ Lectures ■ Office Hours

USC Viterbi School of Engineering 7

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Unsolicited Advice

- This is a tough Class...
 - Structured in packets of material
 - Study regularly each subject
- We are here to Help
 - Just drop by the office during any of my office hours
 - Whenever I'm around
- Do not Cheat!
 - I get upset (that is not good!)
 - Later you might need a letter of reference from me...

USC Viterbi School of Engineering 8

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Compilers

- What is a Compiler?

USC Viterbi School of Engineering 9

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Compilers

- 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?

USC Viterbi School of Engineering 10

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Compilers

- 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

USC Viterbi School of Engineering 11

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Compilers

- 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)
 - which are then interpreted
 - Or a hybrid strategy is used
 - Just-in-time compilation

USC Viterbi School of Engineering 12

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Taking a Broader View

- Compiler Technology = Off-Line Processing
 - 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
- Examples
 - Macro expansion
 - PL/I macro facility — 10x improvement with compilation

13

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Taking a Broader View

- Compiler Technology = Off-Line Processing
 - 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
- Examples
 - Macro expansion
 - PL/I macro facility — 10x improvement with compilation
 - Database query optimization

14

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Taking a Broader View

- Compiler Technology = Off-Line Processing
 - 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
- Examples
 - Macro expansion
 - PL/I macro facility — 10x improvement with compilation
 - Database query optimization
 - Emulation acceleration
 - TransMeta “code morphing”

15

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

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
- Writing a compiler exposes practical algorithmic & engineering issues
 - Approximating hard problems; efficiency & scalability

16

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Intrinsic Interest

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

<i>Artificial intelligence</i>	Greedy algorithms Heuristic search techniques
<i>Algorithms</i>	Graph algorithms, union-find Dynamic programming
<i>Theory</i>	DFAs & PDAs, pattern matching Fixed-point algorithms
<i>Systems</i>	Allocation & naming, Synchronization, locality
<i>Architecture</i>	Pipeline & hierarchy management Instruction set use

17

USC Viterbi School of Engineering CSCI 565 - Compiler Design Spring 2010

Intrinsic Merit

➤ Compiler construction poses challenging and interesting problems:

- Compilers must do a lot but also run fast
- Compilers have primary 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

18

About the Instructor

- My own research
 - Compiling for Advanced Architectures Systems
 - Optimization for Embedded Systems (*space, power, speed*)
 - Program Analysis and Optimization
 - Reliability and Distributed Embedded Systems
 - Rethinking the fundamental structure of optimizing compilers
- Thus, my interests lie in
 - Interplay between Compiler and Architecture
 - Static Analysis to discern Program Behavior
 - Run-time Performance Analysis