Overview of the Class

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.diniz@tagus.ist.utl.pt)
- Office Hours: Tuesdays,Thurdays 1.30 PM, 2-N9.13
- Textbook: **"The Dragon Book" not required, but helpful.**
- Web Site: http://webdei.rnl.ist.utl.pt/~ped/Teaching/2009/Compilers
  - Projects, homework, slides…
  - I will not have handouts in class; get them from the web
  - The On-line Forum is very important.

Basics of Grading

- Exams
  - Midterm (in class) 20%
  - Final (in class) 20%
- Homeworks (5/6) 30%
- Programming Projects
  - Developing Your Own Compiler/Language
    - Lexical Analysis 5%
    - Syntactic 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,…)
- Forum Participation 2% bonus!

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*)

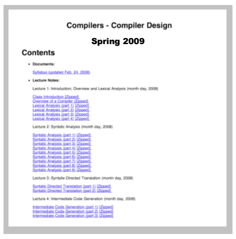
Lecture 1
Lecture 2
Lecture 3 ← HW 1
Lecture 4 ← Project 1
Lecture 5
Lecture 6
Lecture 7 ← HW 2
Lecture 8 ← Project 2
Lecture 9 ← HW 3
Lecture 10 ← Project 3

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*)

On-line Material for the Class

- Class Web Site
  - http://web.ist.utl.pt/~pedro.diniz/Teaching/2009/Compilers


Compilers - Compiler Design

Spring 2009

Contents

- **Documents:**

  Syllabus (updated Feb. 24, 2008)

- **Lecture Notes:**

  Lecture 1: Introduction; Overview and Lexical Analysis (month day, 2008)

  Class Introduction [Zipped]
  Overview of a Compiler [Zipped]
  Lexical Analysis (part 1) [Zipped]
  Lexical Analysis (part 2) [Zipped]
  Lexical Analysis (part 3) [Zipped]
  Lexical Analysis (part 4) [Zipped]

  Lecture 2: Syntatic Analysis (month day, 2008)

  Syntatic Analysis (part 1) [Zipped]
  Syntatic Analysis (part 2) [Zipped]
  Syntatic Analysis (part 3) [Zipped]
  Syntatic Analysis (part 4) [Zipped]
  Syntatic Analysis (part 5) [Zipped]
  Syntatic Analysis (part 6) [Zipped]
  Syntatic Analysis (part 7) [Zipped]
  Syntatic Analysis (part 8) [Zipped]
  Syntatic Analysis (part 9) [Zipped]

  Lecture 3: Syntatix-Directed Translation (montn day, 2008)

  Syntatic Directed Translation (part 1) [Zipped]
  Syntatic Directed Translation (part 2) [Zipped]

  Lecture 4: Intermediate Code Generation (month day, 2008)

  Intermediate Code Generation (part 1) [Zipped]
  Intermediate Code Generation (part 2) [Zipped]
  Intermediate Code Generation (part 3) [Zipped]

Schedule of the Class

LEIC 2º ano	Segunda-feira	Terça-feira	Quarta-feira	Quinta-feira	Sexta-feira
8:00 - 8:30					
8:30 - 9:00					
9:00 - 9:30					
9:30 - 10:00			Comp – 2N9.13 Office Hours Pedro Diniz		
10:00 - 10:30					
10:30 - 11:00					
11:00 - 11:30					
11:30 - 12:00	Comp - 1.17 Practical Class Jan Cederquist		Comp - 0.15 Practical Class Pedro Diniz		
12:00 - 12:30					
12:30 - 13:00				Comp – 2N9.13 Office Hours Pedro Diniz	
13:00 - 13:30					
13:30 - 14:00					
14:00 - 14:30	Comp – 2N3.1 Office Hours João Sacramento				
14:30 - 15:00		Comp - A1 Lectures Pedro Diniz		Comp - A1 Lectures Pedro Diniz	
15:00 - 15:30					
15:30 - 16:00					
16:00 - 16:30		Comp - 2N9.13 Office Hours Pedro Diniz			
16:30 - 17:00					
17:00 - 17:30			Comp - 1.1 Practical Class Jan Cederquist		
17:30 - 18:00		Comp - 0.15 Practical Class Pedro Diniz			
18:00 - 18:30					
18:30 - 19:00					
19:00 - 19:30					
19:30 - 20:00					
20:00 - 20:30					

🟩 Lectures 🟦 Practical Class 🟩 Office Hours

Compilers

- What is a Compiler?

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?

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

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

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

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

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"

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

Intrinsic Interest

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

Artificial intelligence	Greedy algorithms Heuristic search techniques
Algorithms	Graph algorithms, union-find 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 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

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