Overview of the Class

Copyright 2017, 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.
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)

- Lectures: Friday, 1.00 – 4.20 PM, KAP 163

- Textbook: “Engineering a Compiler” by Cooper and Torczon

- Web Site: http://www.isi.edu/~pedro/Teaching/CSCI565-Spring17
  - Projects, homework, slides, sample solved exercises …
  - I will not have handouts in class; get them from the web in electronic format
Basics of Grading

• Exams
  – Midterm (aka first test, in class) 20%
  – Final (aka second test, in class) 25%

• Homeworks (4) 20%

• Programming Projects
  – Developing Your Compiler Analyses and Passes in LLVM
    • Control-Flow Analysis 10%
    • Register Allocation 25%
  – Website has a lot of info and code!
    • How to Get Started.
    • Links to LLVM
    • Basic Data Structures (linked-list, arrays, graphs, …)
Homeworks & Projects

- **Homeworks**
  - Individual
  - Designed to prepare you for the Midterm and Final exams

- **Projects**
  - Use the LLVM infrastructure to perform simple scalar analysis. Code generation for a simplified set of constructs targeting the MIPS architecture (or another ISA you suggest) and preforming some limited register allocation. Use of an existing MIPS software simulator for validation of your outputs.
  - Individual effort
  - Some programming but plenty of support from TA.
  - Plenty of auxiliary code basis so that you can focus on the concepts
  - You can develop it on your machine but need to demo on UNIX @ USC
  - Increasing difficulty at Grade Weight
    - First project focus is simpler – hence less weight
    - Second project focus on code generation. You can use a simulator we provide to validate your compilation.
<table>
<thead>
<tr>
<th>Tentative Syllabus</th>
<th>Lecture 1</th>
<th>Lecture 2</th>
<th>Lecture 3</th>
<th>Lecture 4</th>
<th>HW 1</th>
<th>Lecture 5</th>
<th>HW 2</th>
<th>HW 2</th>
<th>HW 2</th>
<th>HW 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Introduction &amp; Overview</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Lexical Analysis: Scanning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Syntactic Analysis: Parsing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Syntax-Directed Translation &amp; Parse Tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Intermediate Code Generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Control-Flow Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Semantic Analysis and Error Checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Run-Time Environment &amp; Storage Organization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Data-Flow Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Code Generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Instruction Scheduling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Register Allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• More Optimizations (time permitting)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pedro Diniz  
pedro@isi.edu
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 (but not much)
  – Projects are graded on functionality, documentation, and lab reports more than style (*results do matter*)
On-line Material for the Class

- Class Web Site
  - http://www.isi.edu/~pedro/Teaching/CSCI565-Spring17

---

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: Syntactic Analysis (month day, 2008)
    - Syntactic Analysis (part 1) [Zipped]
    - Syntactic Analysis (part 2) [Zipped]
    - Syntactic Analysis (part 3) [Zipped]
    - Syntactic Analysis (part 4) [Zipped]
    - Syntactic Analysis (part 5) [Zipped]
    - Syntactic Analysis (part 6) [Zipped]
    - Syntactic Analysis (part 7) [Zipped]
    - Syntactic Analysis (part 8) [Zipped]
  - Lecture 3: Syntactic-Directed Translation (month day, 2008)
    - Syntactic Directed Translation (part 1) [Zipped]
    - Syntactic 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]

- Class Forum on Piazza
## Schedule of the Class

<table>
<thead>
<tr>
<th>USC Spring 2010</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00 - 8:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8:30 - 9:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:00 - 9:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:30 - 10:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:00 - 10:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:30 - 11:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:00 - 11:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:30 - 12:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:00 - 12:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:30 - 13:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:00 - 13:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:30 - 14:00</td>
<td>Lectures</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:00 - 14:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:30 - 15:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:00 - 15:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:30 - 16:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:00 - 16:30</td>
<td></td>
<td></td>
<td></td>
<td>Office Hours</td>
<td></td>
</tr>
<tr>
<td>16:30 - 17:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17:00 - 17:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17:30 - 18:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18:00 - 18:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18:30 - 19:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:00 - 19:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30 - 20:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20:00 - 20:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lectures: KAP 163
Office Hours: PHE 514

### Lectures

### Office Hours
Unsolicited Advise

• 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…
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

<table>
<thead>
<tr>
<th>Artificial intelligence</th>
<th>Greedy algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heuristic search techniques</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Graph algorithms, union-find</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dynamic programming</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theory</th>
<th>DFAs &amp; PDAs, pattern matching</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed-point algorithms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Systems</th>
<th>Allocation &amp; naming,</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Synchronization, locality</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Pipeline &amp; hierarchy management</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Instruction set use</td>
</tr>
</tbody>
</table>
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/verify Program Behavior
  – High-Performance, Reconfigurable and Configuration Computing
  – Resilience Computing