Overview of the Class

Critical Facts

Compilers —*Compiler Design and Implementation*

This course is intended to give the students a thorough knowledge of compiler design techniques and tools for modern computer programming languages. This course covers advanced topics such as data-flow analysis and control-flow analysis, code generation and program analysis and optimization.

- **Key Objectives:**
  - Understanding programming languages' compilation phases, in particular for imperative and object-oriented (OO) languages;
  - Design and specify the syntax and semantics of a programming language;
  - Understand and using data structures and key used algorithms and their trade-offs in compiler implementation;
  - Building a compiler or productivity tools based on compiler and programming language concepts

Learning Outcomes & Competences

- Develop and implement software processing systems that rely on high-level programming languages concepts and can be implemented on contemporary processing hardware

- Master key concepts in existing compilation and productivity tools, namely:
  - Lexical analysis (regular expressions and finite automata)
  - Syntactic analysis (CFGs and PDAs)
  - Semantic analysis
  - Code "optimization"
  - Code generation techniques targeting contemporary processors or virtual machines

Prerequisites

Students should be familiar with and/or have strong:

- Concepts of theory of computation (*e.g.*, regular sets, or CFGs);
- Design and analysis of algorithms (*e.g.*, asymptotic complexity, divide and conquer and dynamic-programming techniques);
- Programming skills using dynamic, pointer-based data structures in C, C++ or Java programming languages.
- Basic concepts of imperative programming languages such as scoping rules, parameter passing disciplines and recursion.

Reference Textbooks

- **Main:**
  - A. Aho, M. Lam, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd Ed., Addison Wesley, 2007. ISBN: 0321486811
  - Appel, Andrew Wilson, *Modern Compiler Implementation in Java*, 2nd Ed., Cambridge Univ. Press, 2002. ISBN: 0-521-82060-X
- **Complementary:**
  - K. Cooper and L. Torczon, *Engineering a Compiler*, Morgan Kaufmann, 2nd ed., 2011. ISBN: 012088478X
  - Torben Ægidius Mogensen, Introduction to Compiler Design, Second Ed., Undergraduate Topics in Computer Science, Springer 2017, ISBN 978-3-319-66965-6.
- **Advanced Topics:**
  - Muchnick, Steven, *Advanced Compiler Design and Implementation*, Morgan Kaufman Pubs., 1997. ISBN 1-55860-320-4
  - Allen, Randy; and Kennedy, Ken, *Optimizing Compilers for Modern Architectures*, Morgan Kauffman Pubs.,2001 ISBN 1-55860-286-0

Tentative Syllabus

- Introduction & Overview
- Lexical Analysis: Scanning
- Syntactic Analysis: Parsing
- Symbol Tables and Semantic Analysis
- Type Checking and Error Checking
- Intermediate Code Generation
- Run-Time Environment & Storage Organization ← First Test
- Control-Flow Analysis
- Data-Flow Analysis
- Code Generation
- Instruction Scheduling
- Register Allocation
- More Optimizations (*time permitting*) ← Second Test

Faculty and Lectures

- Instructors:
  - Prof. Pedro C. Diniz (FEUP, room I137 - pedrodiniz@fe.up.pt)
  - Prof. Mário Florido (FCUP, room TBD - amflorid@fc.up.pt)
  - Prof. João Bispo (FEUP, room J204 jbispo@fe.up.pt)
  - Dr. Tiago Carvalho (FEUP, room J204, tdrc@fe.up.pt)
  - Lázaro Costa (FEUP, room: J204, lazaro@fe.up.pt)
  - Pedro Pinto (FEUP, room J204, p.pinto@fe.up.pt)
- Lectures:
  - Thursday, 10.30 AM – 12.30 PM, B.013
  - Thursday, 2.30 PM – 4.30 PM, B.035
- Office Hours:
  - Thursday, 1 hour just before class, room: I.137

Basics of Grading

- Tests
  - First Test (individual, min. 8/20) 25%
  - Second Test (individual, min. 8/20) 25%
  - Continuous Evaluation Grade (AC) is average of both tests
  - You may bring an A4 sheet with your notes (both sides)
- Programming Project 50%
  - Group project but individual grades (min. 8/20)
  - Multiple check-points with specific grade weights (see next slides)
- **Warning: Compilers is a Holistic Class**
  - You Need to grasp concepts across multiple topics
  - Exams (AC) and project grade (PRJ) cannot differ by more than 4 points
    - If so, higher grade is "truncated" to lower grade + 4.0.
    - Average computed with adjusted grades (50% AC, 50% PRJ)
  - **Bottom line:** Don't neglect neither the theoretical material nor the project

Tests

- Written and **Individual**
  - You may bring an A4 sheet with your notes (both sides)
- 2-hours long and **Exclusive**
  - First one mid-term focusing on first part of class material
  - Second one at the end of classes focusing on the remainder material
  - Either or both can be subject to a make-up test (after the end of classes)
- Based on the vast collection of Sample Exercises
  - Periodically posted at the class web site
  - With solutions and comments
  - Major source of inspiration for tests (*i.e.*, there is no sample test)
  - You can review them with lecturers during Practical Classes or Labs

Programming Project

- Developing a Compiler for a subset of Java
  - Challenging large-scale programming project
    - Covers a lot of the concepts in the lectures
    - You need to understand concepts and make implementation choices
  - Team Effort (4 elements per group)
    - You need to be organized and clear about who does what.
  - Project Support Material at the class web site
    - Links to Basic Data Structures (linked-list, arrays, graphs,…)
    - Show up for the Labs so that Lectures can help you
- Grading: Checkpoints & Presentation
  - First Verification Checkpoint: 5%
  - Second Verification Checkpoint : 10%
  - Third Verification Checkpoint : 10%
  - Final Submission and Checking (against battery of tests): 60%
  - Project Presentation and Discussion: 15%

Class-Taking Techniques

- 2-hour Lectures (Lecture or Theory Class):
  - Presentation of the topics, exercises related to compiler theory and practice
  - Discussions of ideas, solutions, etc.
- 2-hour Practical Classes or Labs (TPs):
  - Resolution and discussion of topics related to the project
  - Meeting with instructors
  - **Exercises about specific topics**

Class-Taking Techniques (cont.)

- We will use projected material extensively
  - We 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
  - Check and review the sample exercises and solutions
- Compilers is ***also*** a programming course
  - Projects are graded on functionality, documentation, and lab reports more than style (*results do matter*)

On-line Material for the Class

- SIGARRA
  - Organization of the course
  - Slides
  - Class material like sampled Exercises with Solutions
  - Previous Year's Eaxms
  - MS Teams
  - https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=484379

Schedule of the Class

COMP Spring 2022	Monday	Tuesday		Wednesday				Thursday	Friday
08:00-08:30									
08:30-09:00		Practice Pinto B204		Practice Carvalho B205	Practice Diniz B202	Practice Costa B312	Practice Pinto B310		
09:00-09:30									
09:30-10:00									
10:00-10:30									
10:30-11:00		Practice Florido B342	Practice Bispo B302	Practice Costa B217	Practice Diniz B207			Theory Diniz/Florido B013	
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								Theory Diniz/Florido B035	
14:30-15:00									
15:00-15:30									
15:30-16:00									
16:00-16:30									
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									

Lectures (Theory/Practice)
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!
  - We get upset (that is not good!)
  - Later you might need a letter of reference from us…

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
  - 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 Instructors

- Our 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, our Interests lie in
  - Interplay between Compiler and Architecture
  - Static Analysis to discern/verify Program Behavior
  - High-Performance, Reconfigurable and Configuration Computing
  - Resilience Computing