Instructor: Dr. Pedro Diniz, email pedro@isi.edu
Lectures: Wednesday, 3.30 – 6.20 PM, RTH 115, phone: (213) 740 4518 
Office hours: Wednesday, 2.30 to 3.30 PM (just before class), Office: SAL 234

Teaching Assistants: TBD.

Description: 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 (not covered in CSCI 410x - Translation of Programming Languages) such as data-flow analysis and control-flow analysis, code generation and program analysis and optimization.

Prerequisites: Students should be familiar with the concepts in theory of computation (e.g., regular sets and context-free grammars); design and analysis of algorithms (e.g., asymptotic complexity, divide and conquer and dynamic-programming techniques); and have strong programming skills using dynamic, pointer-based data structures either in C or C++ programming languages. Students should also be familiar with basic concepts of imperative programming languages such as scoping rules, parameter passing disciplines and recursion.

Prerequisites include the USC courses listed below. If you do not meet these requisites please contact the instructor to determine if you should take the class.

- CSCI 455x (Introduction to Programming Systems).
- CSCI301 - Theory of Computation or CSCI430 - Automata Theory
- CSCI303 - Design and Analysis of Algorithms
- CSCI410 - Translation
- CSCI420 - Concepts in Programming Languages

Assignments: We expect to have 5 individual homework assignments throughout the course in addition to a Mid-Term exam and a Final in-class exam. Homeworks are due at the beginning of the class on the due date. Homeworks are organized and structured as preparation for the midterm and final exam, and are meant to be a studying material for both exams.

*Late homeworks will not be accepted* as the solutions are posted within hours of the end of the class they are due. You need to turn in a signed hardcopy of your homeworks (exception for remote students). In extreme circumstances please ask a fellow student to turn in your homework in a sealed envelope.

There will also be 3 individual programming projects in this class. The goal of these projects is to have the students experiment with very practical aspects of compiler construction and program analysis. These projects will cover topics we
study in class and with the vast code basis available at the class web site you should have no problem completing each of them in under a week.

Tentatively, this year’s project focus on the following topics:

1. Type checking and type inference
2. Intermediate code generation
3. Control-flow analysis and optimization

For each project we will have a specific set of test programs some of which we make available to you before hand for testing. We use additional test programs for our own evaluation and whose output will be compared against your project’s outputs. At the end of each programming assignment we will make available an implementation that you might want to use for the following programming project in case there might be dependences between projects.

**Grading:**

We will assign 25% of this class grade to homeworks (some might have larger percentages due to difficulty), 25% for the programming projects (10% first, 15% second), 20% for the mid-term and 30% for the final exam. The Final exam is not comprehensive.

**Important Note on Grading:**

It is absolutely against university policy for faculty to grant appeals for grade change for any reasons other than mistakes in grading. The student's request is completely inappropriate and - even if faculty is sympathetic - it is not possible within the rules to grant the requested change.
The recommended textbook for this class is the first volume listed below. I’ll draw most of the material from this textbook but will occasionally draw some material from the other references. Occasionally, I’ll distribute supplemental materials in class to cover topics that are not easy to find elsewhere.

Title: *Compilers: Principles, Techniques and Tools*
Authors: A. Aho, M. Lam, R. Sethi and J. Ullman
Publisher: Addison-Wesley 2nd edition
Year: 2007

Title: *Engineering a Compiler*
Authors: Keith Cooper and Linda Torczon,
Publisher: Morgan-Kaufman Publishers

Title: *Advanced Compiler Design and Implementation*
Author: S. Muchnick,
Publisher: Morgan-Kaufmann Publishers

Class Material: Please gain access to course-related material on-line at the class web site (http://www.isi.edu/~pedro/Teaching/CSCI565-Spring11/) or using the Distance Education Network (DEN) Blackboard facility at den.usc.edu. All students must register with DEN on website before gaining class web-page access.

Do not mail questions to the TA or instructor. Instead, post all questions on DEN’s Blackboard. Your questions will be answered as promptly as possible, and typically within 24 hours after being posted.

Lectures: Below is a tentative description of the contents of each lecture excluding the midterm and final lectures. We may change the order to accommodate the materials you need for the projects. The first two lectures are meant to be review of material you have covered elsewhere. As such the pace is going to be higher than normal. *Do not panic. We are here to help you.*

*Lecture 1: Introduction; Lexical Analysis, Regular Languages and Finite Automata*
- Introduction to compilation and programming languages
- Typical compiler structure and its internal phases
- Lexical Analysis: Its goal and input/output
- Finite Automata Theory and Regular Expression.
- Automatic Recognition of REG (example of Lex/Flex)
- Limitations of Regular Expressions.

*Lecture 2: Syntactic Analysis, Context-Free Languages*
- Syntactic Analysis: Its goal and input/output
- Context Free Languages and PDA.
- Parsing Algorithms for LL and LR Grammars; Recursive Descendent and Shift-Reduce Techniques
- Other Parsing algorithms: CYK
- Automatic Recognition of CFL (example of Bison/Yacc)
- Ambiguity and how to avoid it.
Lecture 3: Semantic Analysis and Syntax-Directed Translation
- Limitations of Syntactic Analysis.
- Type Checking.
- S-Attributed Grammar and L-Attributed Definitions.
- Translation to Syntax Trees and DAGs.
- Evaluation Schemes.

Lecture 4: Intermediate Representation and Intermediate Code Generation
- Introduction to Intermediate Code Generation, High-level Overview
- Intermediate Representation: Three-Address Instructions
- Syntax-Directed Translation Schemes for Code Generation
- Code for Expressions, Assignment, and Arrays
- Boolean and Relational Operators, Conditionals, & Control flow
- Backpatching

Lecture 5: The Procedure Abstraction, Run-Time Environment and Storage Allocation
- Introduction, discussion of control abstraction, name spaces, and external interfaces
- Name Spaces & Symbol Tables.
- Symbol Tables & Storage Layout
- Allocating Storage and Establishing Addressability
- Run-time Structures for Object-oriented Languages
- Finishing up Run-time Structures for OOLs.

Lecture 6: Code Generation and Instruction Selection
- Introduction to Code Generation, High-level Overview
- Basic Blocks
- Code for Expressions, Assignment, and Arrays in DAGs and Basic Blocks
- Boolean and Relational Operators, Conditionals, & Control flow
- Procedure Calls & Dispatch, again

Lecture 7: Register Allocation
- The Importance of Register Allocation and Assignment
- Top-Down Simple Allocation Algorithm and Its Limitations
- Global Register Allocation via Graph Coloring

Lecture 8: Introduction to Optimization
- Control-Flow Analysis: Dominators and Post-Dominators
- Finding Loops and Loop Invariant Code
- Strength Reduction
- Constant Propagation and Constant Folding
- Basic Induction Variable Recognition

Lecture 9: Data-Flow Analysis
- What is Data-Flow Analysis?
- Reaching Definitions and Live Variables Analysis
- Forward and Backwards Problems
- Available Expressions
- Generalization of Data-Flow Analysis

Advanced Topics

Lecture 10: Instruction Scheduling
- Local Instruction Scheduling; List-scheduling algorithm
- Architectural complications
- Beyond basic-blocks

Lecture 11: Transformations for the Memory Hierarchy
- Introduction and Rationale: Registers versus Caches
- Loops and Locality: Temporal and Spatial Locality
- Scalar Replacement and Register Pressure
- Loop unrolling, Unroll-and-Jam and Tiling in Scientific Codes
- Case Study: Matrix-vector and Matrix-vector Multiplication