CS 375: Compilers

Spring 2015: TTh 12:30 - 2:00 in GDC 1.304, Unique No. 52010.

Instructor: Gordon S. Novak Jr., GDC 3.824; Office Hours: TTh 3:30 - 5:00 PM

TA: Akanksha Bansal, akankshabansal90@gmail.com Office Hours: TBD in the GDC basement TA station, and by appointment.

Prerequisites: CS 314 and CS 429. Recommended: CS 439 and CS 345.

Optional Text: Aho, Lam, Sethi, & Ullman, *Compilers: Principles, Techniques, and Tools*

Course Notes: Strongly recommended. Available in GSB 3.136 . Available on-line by **Contents** or Index or PDF. Also see the Vocabulary.

iClicker: Each student is required to buy/rent an iClicker (any version of iClicker device; not iClicker GO app). This will be used for attendance and to reinforce and practice with the class material. Two iClicker points are given just for voting, and an additional point is given for a correct answer. Most iClicker questions and answers are online at **Clicker Questions**, and it is okay to review them in advance. The clicker scores will be converted to a grade by making the highest student score at least 108 and linearly scaling other scores; this gives some extra points to account for minor illness, forgotten or malfunctioning clicker, etc. Adjustments to clicker scores will only be made for significant causes such as major illness. Bringing another student's clicker to class is considered to be cheating.

Register your iClicker Use your **UT EID** as the "Student ID".

Web Page: http://www.cs.utexas.edu/users/novak/cs375.html

Program Directory: /projects/cs375/ on CS Linux machines.

FTP Directory: ftp://ftp.cs.utexas.edu/pub/novak/cs375/

Course Description:

CS 375 covers the design of Compilers, which translate programming languages that are easy for humans to use (Java, C++, etc.) into the difficult-to-understand machine language that is executed by computer hardware. Because machine language is the *only* language that can actually be executed, the compiler, along with the operating system, is one of the central pieces of systems software that makes computers usable.

This course will cover the full range of compiler topics. Each student will write a real compiler for most of the Pascal programming language, producing machine code that we will run on hardware. This compiler is an excellent capstone project for a degree in Computer Science: it is a large project that produces an industrial-scale software product. Algorithms, Data Structures, Programming Languages, Architecture, and Theory are combined in a compiler, so this course brings together the courses of the undergraduate CS curriculum into a coherent whole.

By the end of the course, the student will have completed a significant rite of passage and have the confidence of having written a major component of systems software.

The course covers the major parts of a compiler, in the order in which they operate in the compiler itself:

- **Lexical Analyzer:** The Lexical Analyzer reads characters from the input file and groups the characters into words or *tokens* in internal form: keywords, identifiers, numbers, operators.
- **Parser:** The Parser receives the tokens from the lexical analyzer and combines them into structured sentences in the form of *abstract syntax trees* or *ASTs*.
- **Semantics:** Semantic processing includes type checking and generation of code to access data structures such as arrays and records.
- **Optimization:** The efficiency of the code produced by the compiler can be dramatically improved by optimization. The optimizer makes use of concepts from CS theory such as graph theory and set theory.
- **Code Generation:** This is the final stage, in which actual machine code is generated. Theory is used here too, e.g. graph coloring to map the program onto machine resources such as registers.
- **Advanced Topics:** The course will include the topics of object-oriented programming, partial evaluation, translation between programming languages, pattern matching, just-in-time compilation, Lisp, and parsing English.

Readings in Aho, Lam, Sethi, and Ullman:

- Introduction: Chapter 1.
- Syntax:
  - Lexical Analysis: Ch. 3 through 3.4.
    - Regular Expressions: Ch. 3.3.3
    - `lex`: 3.5.
  - Grammars
  - Parsing: Ch. 4 through 4.3
    - Operator Precedence Parsing: not in book.
    - Recursive Descent Parsing: 4.4 through p. 220.
    - LR Parsing: 4.5, 4.6.3
    - `yacc`: 4.9.
  - Semantic Analysis
    - Syntax-directed Translation: Ch. 5.1, 5.3
    - Type Checking: Ch. 6.5
  - Error Handling
- Symbol Tables: 2.7, 6.3, 6.5
- Intermediate Code: Ch. 6 - 6.2, 6.4
- Runtime Support: Ch. 7.
- Code Generation: Ch. 8.
- Optimization: Ch. 9

Grading Policies:

Course grades are assigned on the scale A = 93-100, A- = 90-93, B+ = 87-90, B = 83-87, B- = 80-83, etc. provided that the Final Exam grade is at least 65; if the Final Exam grade is below 65, a lower course grade may be assigned at the instructor's discretion. Grades are averaged using the following

weights:

Midterm Exam	20%	Thursday, March 12, in class
Final Exam	30%	Monday, May 18, 9:00-12:00 AM
Clicker Participation	10%	
Programming Assignments:		(10% per day late penalty)
Lexical Analyzer	06%	
Lexical Analyzer using `lex`	04%	
Parser (total of 3 parts)	18%	
Code Generator	12%	

All students must complete all exams and programming assignments. This course has a **very heavy programming load**

Programming projects must be your own individual work. Students may discuss concepts or help with specific problems in another student's code. However, sharing code, working together on program design or flowcharts, or reading someone else's code is not allowed. All code that is given in the class directory may be used as part of your programs.

Programs may be assigned letter grades, which are numerically averaged as A+ = 100, A = 95, A- = 93, B+ = 88, etc. A program that just satisfies the assignment and has no errors will typically be given a grade of A-; grades of A and A+ are given to programs that have something extra, such as extra work (e.g., in optimization) or especially nice code. Note, however, that a grade of A- is averaged several points higher than the bottom of the A range; for example, a student who made 87 on both exams and received A- on all programs would have an average of 90 and receive a grade of A-.